* Structure of Java

In Java program we can create 3 components

- i Class
- ii Interface
- iii enum

To create a Java program wer neen to follow the steps

1. (i) Open an editor like Notepad,
2. (ii) Save the program
3. (iii) Execute the Program.

Note :-
Execution of Java Program is 2 way process

- i] Compile the Program
- ii] Execute the Program

1) class :-

- It is a keyword.
- It is used to create a class block.

syntax to a class block :

```
class class_name {
    { class_ block }.
}
```

enmomonics are Platform dependent.

Examples :-

class shoes{ | class Pen {
} | }

Q] What is Java source file?
Q] What should be the name of file?
Q] What should be the name of extension?
Q] Which components can be created inside Java Program?

Ans

1] A Java Source file is a plain text file containing Java source code & having .Java extension. The .Java extension means that the file is the Java source file. Java source file contain source code for a class, interface & enumerat

2] we should save the file same as class name. It is not mandatory but better for understanding

3] The Java filename require .java extensio

4) we are creating three component in java Program which is class, interface & enumeration.

Declare → create

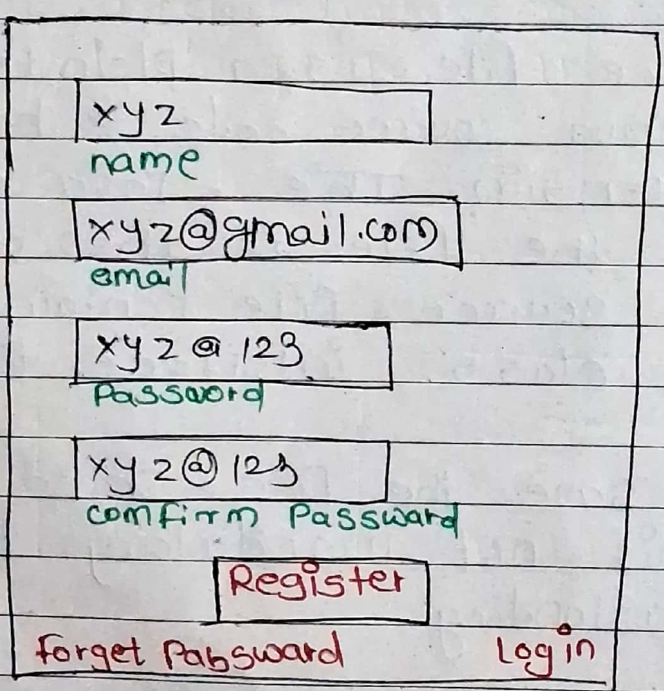
```
class A {

  // Declaration
     Statements

  // initialization
     Statements
}
```

- Declaration Statements → variables, methods, class
- initialization Statements → Static Initializer, Non-static Initialize, constructor

Example :-
Facebook



- name, email, Password, Comfirm Password } variables
- Register, Login, Forgot Password } Action } method

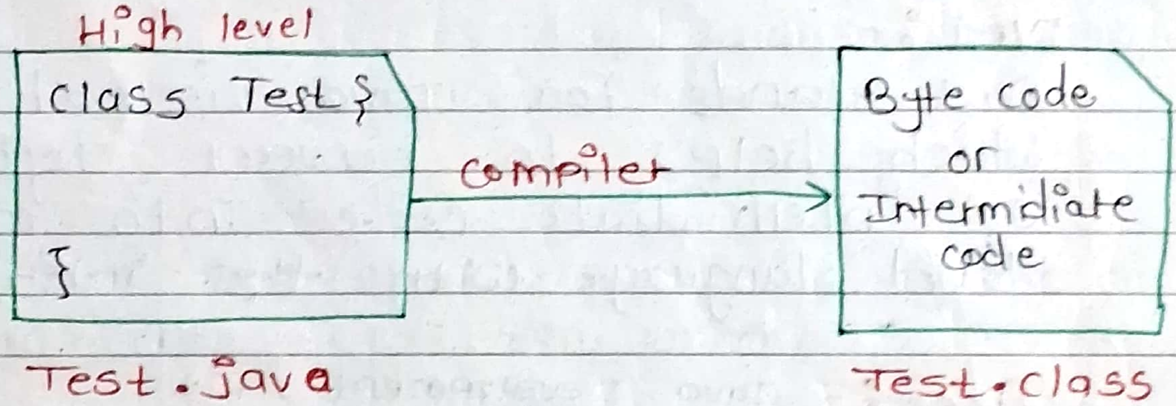
Q) Why Java is a platform independent language?

Java compiler does not directly convert high level language into machine understandable language (binary)

insted of converting this it will convert the code into byte code

byte code also known intermidiate code.

The file is known as class file with the extention .class



Once the class file is ready it can be executed in any of the machines which has JVM (Java virtual machine) in it.

This makes Java as a Platform independent language but JVM is dependent language.

JDK :-
Java Development kit is a Package which consist of devolopment tools like compiler & JRE For the execution.

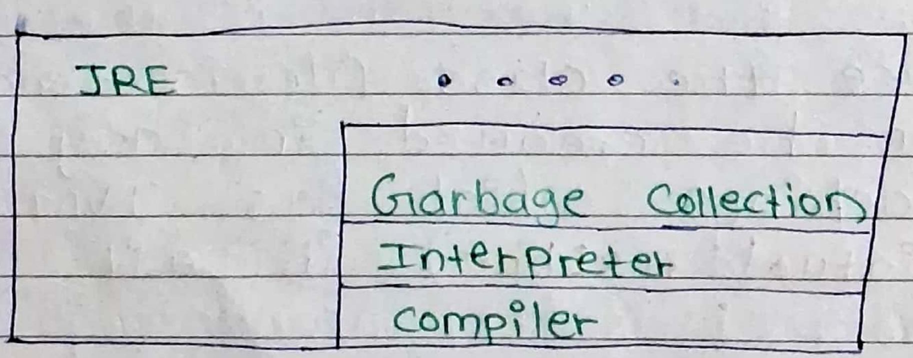
JRE :-

JRE stands for Java Runtime Environment Which is a Environment which consist of Jvm & built-in classes. which is required for the execution of Java Program.

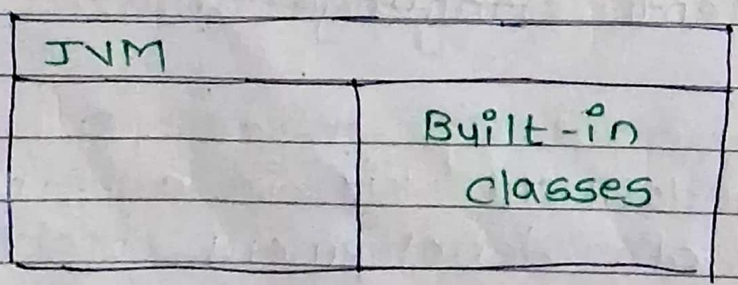
JVM :-

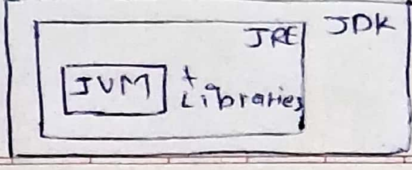
JVM stands for Java virtual machine which helps to convert byte code or intermidiate code into machine level language with the interpreter.

JDK :- Java Development kit.

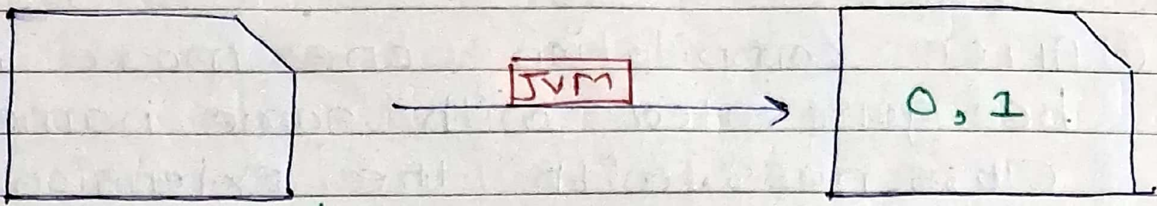


JRE :- Java Runtime Environment





JVM :- Java Virtual Machine



Byte code/ Intermidiate code

machine level language / low level language Binary language

built-in classes :-

These are the classes | program or written by developers

OR

These is a code/ Program which is already Predefined.

* Execution Flow Java Program

① To execute a Java Program create a class.

② Save that file with same as classname.

③ The exeution of Java Program is two-way process.

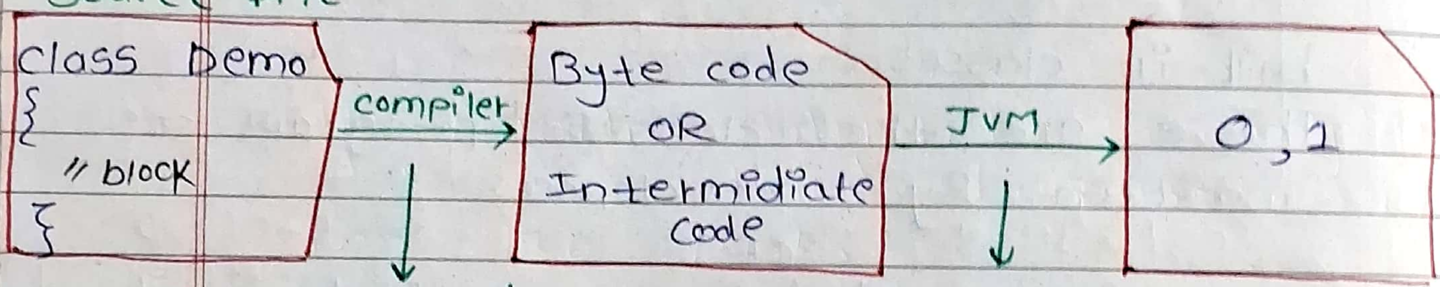
④ First we will compile the Program with help of command

Javac is name of compiler → Javac file-name.java

⑤ After compilation one more file will be generated with same name as Class_name. with the extension .class

⑥ we can execute the program with the help of 'JVM' by using the command
Java class_name.

Source file



name of compiler → Javac filename.java — Command
ex:- javac Demo.java

Command: java filename
ex:- java Demo.

Note :-
we can compile empty class but we cannot excuted.

* Printing statements in Java :-

we have two Printing statements in java

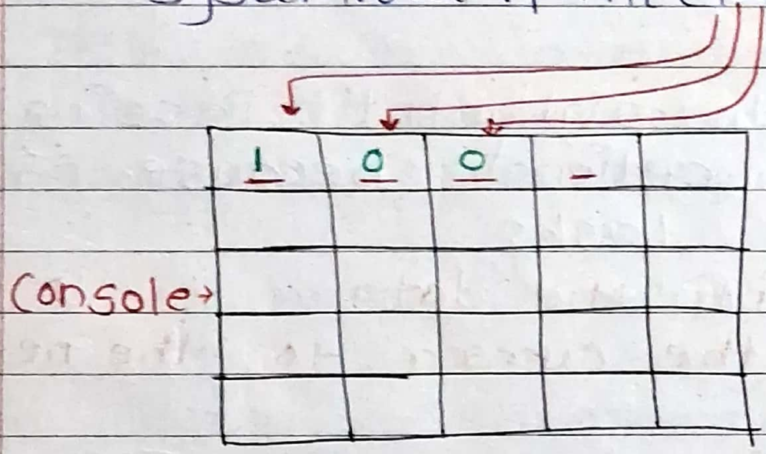
i) System

i> System.out.print(data);

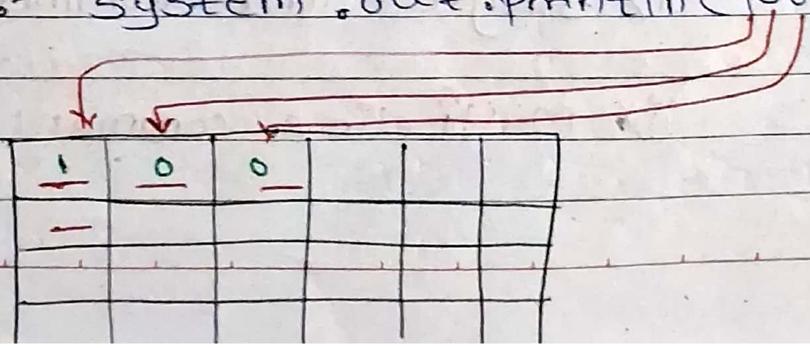
ii> System.out.println( );

System.out.print
This statement is only use to Print the data

Example :-
System.out.print(100); → To print data



System.out.println();
This statement is used to print the data & move cursor to the next line.
Example :- system.out.println(100);



Note :-
Console :-
it is a place where output gets display
In print statement passing the data is mandatory. if we will not pass the data we will get compile time error

System.out.Print( ); → compile time error

In println statement passing the data is optional because println has two tasks
i) To print the data
ii) move the cursor to the nextline

* Comments line :-
This are the line which are ignored by the compiler we

we can use comment in the program by using two ways.
i) single-line comments (//)
ii) Multi-line comments (/* --- */)

Single-line comment
// Program to print 100.

Multiline comment
/* - - - -
- - - -
- - - - - */

* Decision statement :- (condition statement)
This are the statements which is used to implement conditions in the program.

We have 4 decision statements

(i) if statement
(ii) if-else Statement
(iii) ifelse-if ladder
(iv) switch statement

```
if (condition)
{
    // Instruction
}
```

This statement will execute the instruction if the condition is true.

if it is false the control will go out of the block.

if-else statement -

```
if (condition)
{
    // Instruction 1
}
else {
    // instruction 2
}
```

In this statement if the condition is true instruction 1 will execution

if it is false instruction 2 will execute.

else-if ladder statement:

```
if (condition 1) {
   // instruction 1 }
else if (condition 2) {
   // instruction 2 }
else if (condition 3) {
   // Instruction 3 }
 :
else {
   // instruction 4
}
```

Switch (value/variable / expression)
{
case value/ exp : // Instruction 1
case value/ exp : // Instruction 2
case value / exp : // Instruction 3
:
:
default : // Instruction 4.

All keywoords showd be return in lowercase letter.

DATATYPES & SPECIFIES :-

① Type of data stored in a variable or block of memory (Container)

② we have two types of datatype
Primitive
Non-Primitive

Primitive datatype :

① Primitive data type is used to create Primitive variable to store Primitive data

(datatype.max_value),
datatype.min-value);

it specify what type of date PD or standard data we can store inside variable

② Primitive datatype is used to store single value data

what is primitive data

① Number
② character
③ boolean

③ In Java we 8 primitive datatype & this are

Integer:
① Byte (-128 to 127)
② Short (-32768 to 32767)
③ int [-2147483648 to 2147483647]
④ long

Floating point:
⑤ float
⑥ double

Character:
⑦ char

⑧ boolean

Note :-

① Byte, short, int, long are only used to store integers
② float & double are used to store floating point values.
③ Char is used to store characters.
④ Boolean is used to store is Boolean kind of data.

variable = values ; ---- value initialization

$1.4E-45 = -1.4 \times 10^{-45}$

3.402835×10^{35}.

Q) Write a Java Program to store all different types primitive Data?

⇒

```
Class Example {
    public static void main(String[] args)
    {
      int a = 10;
      short
```

```
* Class Example {
    public static void main (String[] args)
    {
      //variable declaration
      int a;
      // variable initialization
      a = 10;
      system.out.println(a);
    }
  }
```

Difference between Print() , Println()

Print()	Println()
① It is used to print the data	It Print the data & move to the cursor next line
② we cannot use print() without passing data	we can use Println() without passing data
③ If we use print() without passing data will get compile time error	If we use Println() without passing data we will not get any error.

* Token :-

① These are the building blocks to construct a java program.

② We have Three Token in Java

(i) keyword
(ii) Identifire
(iii) Literal / constant

① keyword :-

These are the reserved words which have Predefined meaning which is used to perform some specific functionality

class → To create class block

Q) How many keywords we have in Java?
→ 50 + keywords.

Q) List all the keywords in Java?
→
① abstract
② assert
③ boolean
④ break
⑤ byte
⑥ case
⑦ catch
⑧ char
⑨ class
⑩ const
⑪ Continue
⑫ default
⑬ do
⑭ double
⑮ else
⑯ enum
⑰ exports
⑱ extends
⑲ final
⑳ finally
㉑ float
㉒ for
㉓ goto
㉔ i~~z~~f
㉕ implement
㉖ import
㉗ instance of
㉘ int
㉙ interface
㉚ long
㉛ native
㉜ new
㉝ Package
㉞ Private
㉟ Protected
㊱ public
㊲ return
㊳ shot
㊴ Static
㊵ strictfp
㊶ super
㊷ switch
㊸ synchronised
㊹ this
㊺ throw
㊻ throws
㊼ transient
㊽ try
㊾ void
㊿ volatile
(51) while

Rule for keyword :-
All keywords should ~~retu~~ written in lowercase

② Identifier :-

These are the names given to the component of Java like class, variable, methods, etc.

Example :-

```
class classname {
}
```

Here in this Program classname is identifire

* Rules for identifier :-

1> The character allow for identifier are a-z, A-Z, 0-9 $ _

2> Identifier should not sbtart with digit

3> if identifier contains multiple words whitespaces should not be used between.

4> keyword should not be used as identifiers.

Example :-

```
① class A@B
{ → CTE }
```

② class 12AB {
→ CTE
}

③ class mobilephone {
CTE
}

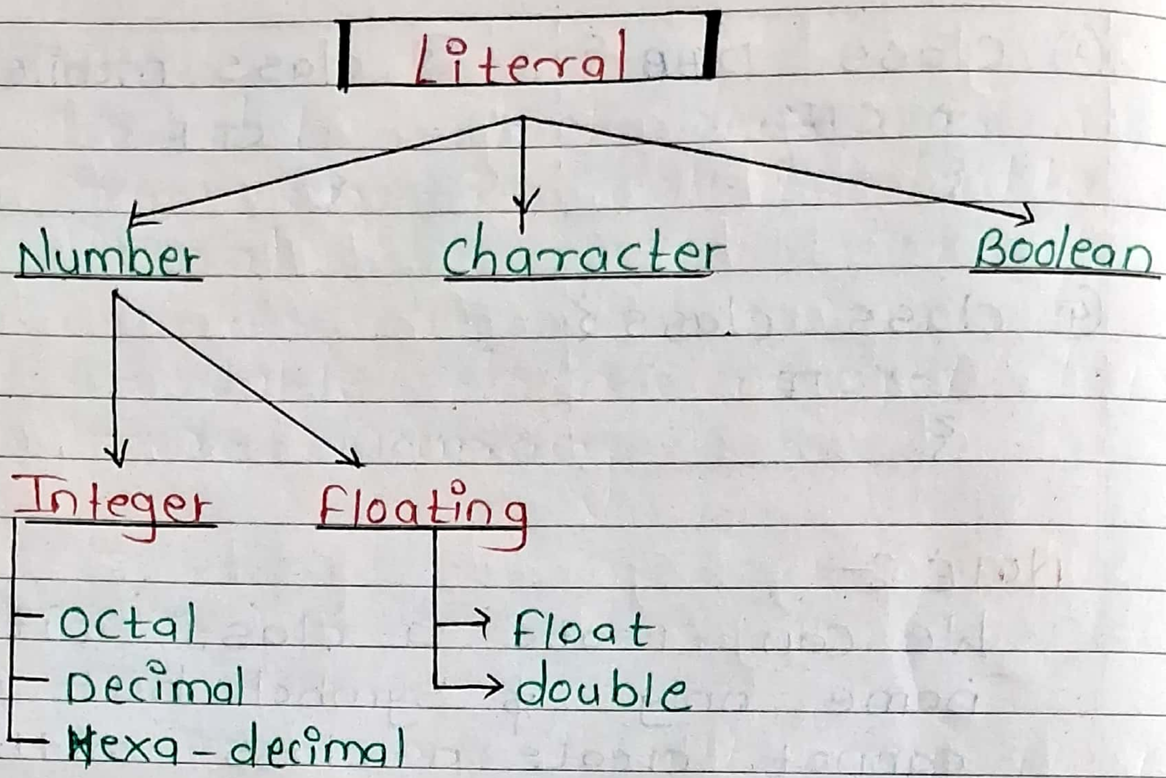
④ class class {
→ CTE
}

NOTE :-
We can create a class with the name only $ symbol but we cannot create a class with only _ symbol

Why?
Because from Java 9th version _ is a keyword and we cannot use keyword as identifier

③ Literal / constant :-
The data stored in a program is called Literal.

There are Three types of Literals
① Number
② Character
③ Boolean.



① Octal Number :-

- The no. which has base value 8 and the range is 0-7

- In java if any no is prefixed with 0 the no. is called as Octal number.

② Decimal Number :-

- The number which base value 10 & range is 0-9

- In Java by default every number is decimal number.

③ Hexadecimal Number :-

The number which has base value 16 & the range is 0-F is called as hexadecimal Number.
Where

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15.

* Floating Point Value :-

1) Float :-
- The data which has less precision is float literal

- In float data we get less accuracy.

2) Double :-

- The data which ~~does~~ has more precision is called double literal

- In double data we get ~~dot~~ more accuracy.

NOTE :-

In Java by default any floating point data is double.

- Precision :-

The number of zeros after decimal Point

* Character :-

- In Java every character must enclose with in single quotes (' ')

- The length of the character must be 1

Program :-

```
class Board {
public static void main (String[] args)
{
    System.out.println ('A')
  }
}
```

* Variables :-

Named block of memory is called as variable

Q. Why we need variable ?
we need ~~be~~ variable to stored the data

Q. How to create variable?
we can create variable with syntax → Datatype identifier;

* Datatypes :-

① datatypes specifies types of data stored in variable or block of memory

② we have two types of datatypes
i) Primitive datatype
ii) Non-primitive datatype

Datatype

- Primitive Datatype
  - byte
  - short
  - int
  - long
  - float
  - double
  - char
  - boolean
- Non-Primitive DataType

① Primitive Datatype :-

① it is used to create Primitive Variable to store data.

② Primitive datatype is used to Store single value data.

Q. what is Primitive data?
① Number
② Character
③ Boolean

In Java, we have 8 Primitive datatype & these are :
① byte
② short
③ int
④ long } integer
⑤ float
⑥ double } floating point (decimal No.)
⑦ char } character
⑧ boolean } boolean value (0 or 1).

* Variable declaration :-
syntax :
datatype identifier 1, identifier 2 ...;
datatype identifier ;

Non-Primitive DataType

Any datatype which is used to store non-primitive data i.e address or Null is called a non-primitive datatype

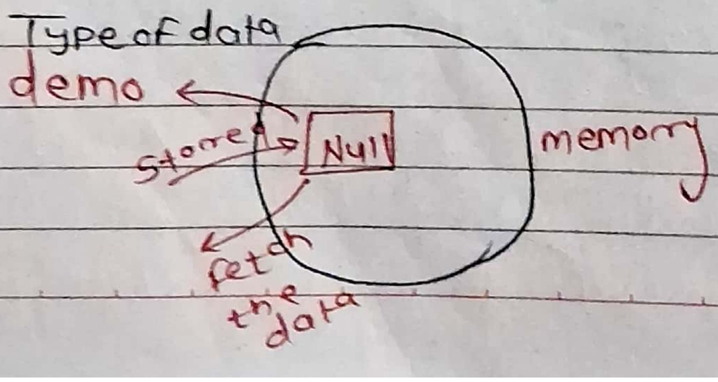
We can store Multi-value data by using non-primitive datatype

Example :-

in Java every class name is non-primitive data type.

```
Class demo {
 public static void main(String [] args)
 {
     demo d;
       d = NULL;
  }
}
```

Here cannot pass single value.



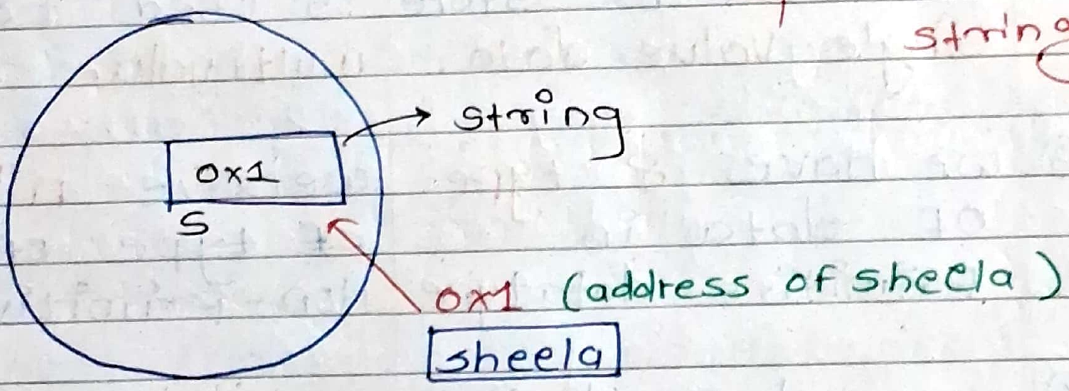
We can store NULL in string also.

NOTE :-
In Java, string is non-primitive datatype.

How to store a string in Java?

```
String s ;
s = "sheela" ;
```

// we use double qoute to store string.



Primitive data is standard data.

Primitive	Non-Primitive
① Primitive DataType are used to ~~Store~~ Create Primitive variables to store primitive data	Non-Primitive datatype are used to create Non-primitive variable used to Store address and NULL
② Primitive datatype are used to store single value data.	Non-primitive datatype is used to store multivalue data.
③ we have 8 type of data in Primitive datatype	wehave n^{th} number ~~of~~ type of data in Non-primitive datatype
④ user defined is not allowed	user defined is allowed

Types of variable

- Based on ~~variable~~ Datatype
  - primitive ~~D~~ variable
  - Non-primitive variable
- Based on scope
  - Local variable
  - static variable
  - Non-static variable

variable are classified into two types based on databtype

i) Primitive variable

- variable created by using primitive datatype to store primitive data is primitive variable.

ii) Non-Primitive variable

Any variable created by using non-primitive datatype used to store address or Null is called as Non-primitive variable.

Based on scope variable are classified into 3 types

i) Local variable
ii) static variable
iii) Non-static variable

i) Local variable :-
Any variable created or declared inside any block other than class block (except class block) is called as local variable

class block / method block

```
class Test { int a ;
    public static void main (string[] args) {
    {
        int b ; // Local variable
    }
}
```

NOTE :-

in above program 'a' is not a local variable because it is created inside class block whereas 'b' is a local variable because it is created inside method block.

* Rules of Local variable :-

① we cannot use local variables without storing the data.

example :-
```
class Test
{
    public static void main (String[] args)
    {
        int b;
        System.out.Println(b); // CTE.
    }
}
```

② We can use the variable only where it is created or declared inside the same block.

Example :-
```
class Test
{
public static void main (String[] args)
{
    int b = 20;
    system.out.println(ab); // CTE
}
public static void m1( )
{
    int a = 10;
}
}
```

compile time error.

③ We cannot declare two variables with same name in the same scope (method) if we will create we will get compile time error.

Example:-

```
class Text
{
  public static void main (String[] args)
  {
      int b = 20;
      char b = 'a';        // CTE
  }
}
```

~~Understanding method~~

* Operators :-

It is a symbol which is used to perform some specific Task or operation

Operand :-
These are the values on which we perform the operation

operands

2 + 3

↑

operator.

- **Purpose of operator :-**
  1. To perform the specific operation
  2. To return the result.

- **Classification of Operator :-**

  1> classification of operator based on operand
  2> Classificat^n of operator based on functionality.

1) classificat^n of operator based on operands :-
we have three types of operator base on number of operands :-

① unary Operator
② Binary operator
③ Ternary operator

classification of operators

- Classificatn of operator based on operands
  - Unary operator
  - Binary operator
  - Ternary operator
- classification of operator Based on functionality
  - ① Arithmetic
  - ② Relational/Comparision
  - ③ Logical
  - ④ Bitwise
  - ⑤ Conditional
  - ⑥ Assignment
  - ⑦ other Special operator

① Unary operator :-

- These are the operators which requires only one operand to execute

- We have two important unary operator :
  ① Increment operator
  ② Decrement operator

① Increment operator :-

- It is used to increase value of variable by 1.

- Symbol is (++)

- we have two types of Increment operator :
  1> Pre - Increment
  2> Post - Increment

1> Pre - Increment :-
It is used to first increase the value then use the value.

Syntax :- ++ variable.

Program :-

```
        class Akshay {
    public static void main(String[] args)
    {
        int a = 10;
        system.out.println(++a);   // 11
    }
}
```

2> Post-Increment :-

It is used to first use the value & then increase value by 1

Syntax :- variable ++

Q1>
```
Int a = 10;
int b = a++ + a++ - ++a - a++ + a++ - a++;
        11    12    13    12    14    14
                   13    (13-1)

System.out.println(a);   // 16
system.out.println(b);   // -6
```

b

a = 10 11 12 13 14 15 (16)

= 10 + 11 - 13 - 13 + 14 - 15
= -6.

② Decrement Operator :-

- It is an operator which is used to decrease the value of variable by 1
- symbol is (--)

- we have two types of decrement operator.

① Pre-decrement.
② Post-decrement.

① Pre-decrement :-
first decrease the value by one then use it.

Syntax :- --variable

Program :-
```
class Akshay {
public static void main(String[] args)
{ int a=10;
System.out.println(--a)     l19
}
}
```

② Post-decrement :-
first use the value & the ~~de~~ decrease the value by 1.

Syntax :- var--

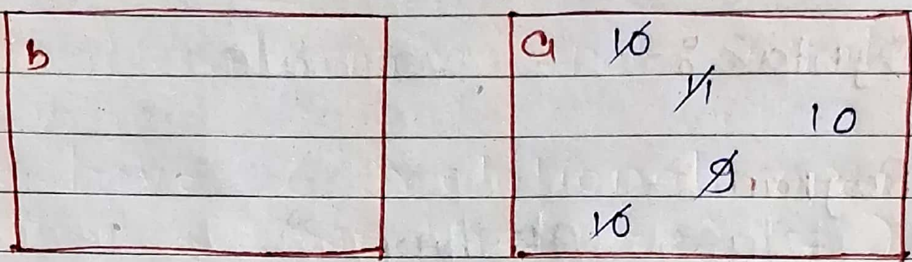
Program :-
```
class Akshay {
Public static void main(String[] args)
{ int a=10;
```

Scanned with OKEN Scanner

```
        System.out.println (a--)    // 10
    }
}
```

Q1>

```
int a=10
int b= a++ - --a + --a + a-- - ++a
System.out.println(a);      // 9
System.out.println (b);     // 9
```



= 10 - 10 + 9 + 9 - 9

Q2>

```
int a=10;
int b = a++ - ++a + --a + a-- + a++ - --a
        20    12    21    21    20    20
```

a=20
b=40

= 20 + 22 + 21 + 21 + 20 - 20
= -2 + 21 + 21 + 20 - 20
= 19 + 21 + 20 - 20
= 60 - 20
= 40

② Binary Operators :-

- These are the operators which requires minimium number of two operands
- we have multiple binary operator based on Functionality.

1) Arithmetic operator :-

(+ , − , * , / , %)

* ↑ Asterisk (*)
* ↑ modulo (%)

e.g :- 10/2 = 5 → Quotient
10 % 2 = 0 → remainder

```
      [5]  → Quotient
   2) 10
    - 10
   ------
     [0]  → remainder.
```

Example :-

```
class Arithmetic {
  Public static void main(String [] args)
  { system.out.Println (10/2); // 5
    system.out.Println (10%2); // 0
  }
}
```

'+' operator acts as a concatenation is any of the operand is string type.

Example :-

1. "om" + 10 = om10
2. 10 + "om" + 2 = 10om20
3. 10 + 20 + "om" = 30om

② Relational/comparisional operator :-

- we have 6 relational operators (>, <, >=, <=, ==, !=)
- Relational operators are binary operators
- The return value of relational operator is boolean.

•

Example :-

```
Class Relational {
  public static void main(string[] args)
  {
    System.out.Println(3>4);    // False
    System.out.Println(5<3);    // False
    System.out.println(3>=3);   // True
    System.out.Println(4<=4);   // True
    System.out.println(3==4);   // False
    System.out.println(7!=7);   // False
  }
}
```

③ Logical operators :-

- we have three logical operators (&&, ||, !)
- logical And (&&), logical OR (||) are binary operator
- ! is a unary operator
- logical operator acts on boolean type of data.

Operand 1	Operand 2	&&	||
True	False	False	True
False	True	False	True
True	True	True	True
False	False	False	False

Example :-

```
class logical {
  public static void main (String[] args) {
    system.out.println(3>4 && 4>8);   // False
    system.out.println (7==7 && 4!=5);   // True
    system.out.println (7!=4 || 7>=8);   // True
  }
}
```

logical not (!) :

! false	=	True
! True	=	false

OP	!
false	True
True	false

(4) Bitwise Operators :-

- we have six bitwise operators (&, |, ~, ^, >>, <<)
  - ~ : Bitwise compliment
  - ^ : Bitwise XOR

- Bitwise compliment(~) is only Unary other are Binary.
- Bitwise operator acts on Bits i.e 0 & 1.

Example :-

5 & 10

first five & 10 will convert into Binary then bitwise & will perform.

10 & 28

2	10	→ 0
2	5	→ 1
2	2	→ 0
	1	↑

1010

2	28	→ 0
2	14	→ 0
2	7	→ 1
2	3	→ 1
	1	↑

11100

Truth Table

Operand 1	Operand 2	& (AND)	| (OR)	^ (XOR)
0	0	0	0	0
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1

```
  0 1 0 1 0
& 1 1 1 0 0
-----------
  0 1 0 0 0   = 8
  16 8 4 2 1
```

(2) 15 & 15

```
  1 1 1 1
& 1 1 1 1
---------
  1 1 1 1   = 15
```

(2) 10 | 10
OR

```
  1 0 1 0
  1 0 1 0
---------
  1 0 1 0   = 10
```

(3) 20 ^ 20

```
  1 0 1 0 0
  1 0 1 0 0
-----------
  0 0 0 0 0   = 0
```

NOTE :-

① If we perform bitwise and (&) operation with the same number we will get same number

② If we perform bitwise or (|) operation with the same number we will get the same number.

③ If we perform bitwise XOR (^) operation with same number we will get output as zero.

④ 35 & 67

	0	1	0	0	0	1	1	
&	1	0	0	0	0	1	1	
	0	0	0	0	0	1	1	= 3

* Storing data in memory :-

1 GB = 1024 MB
1 MB = 1024 KB
1 KB = 1024 byte
1 byte = 8 bits.

To represent one byte of memory
1 byte = 8 bit

left most bit ↓ … ↓ Right most bit.

0	0	0	0	0	0	0	0
	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	64	32	16	8	4	2	1

Example =

① 56 =

2	56	0
2	28	0
2	14	0
2	7	1
2	3	1
	1 ↑	

56   1 1 1 0 0 0

0	0	1	1	1	0	0	0
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^0
	64	32	16	8	4	2	1

= **56**

② 112

64	32	16	8	4	2	1
1	1	1	0	0	0	0

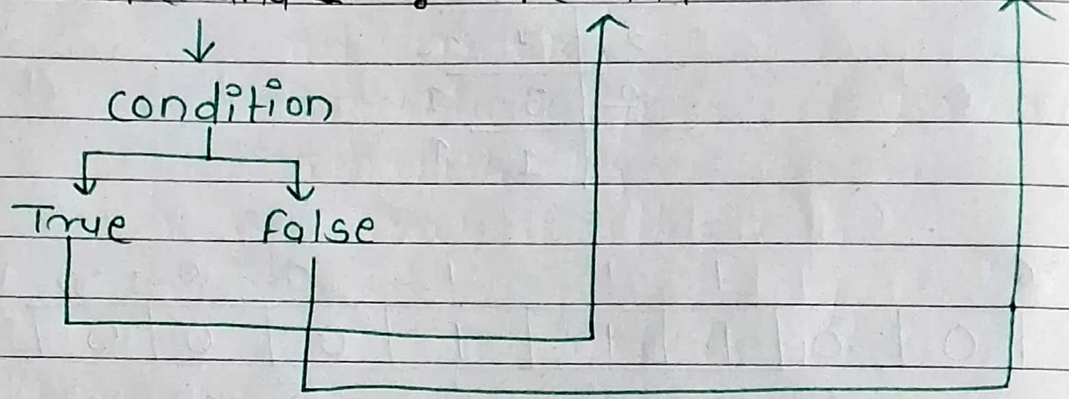
112 → | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

<< → left shift (double)
>> → Right shift (half)

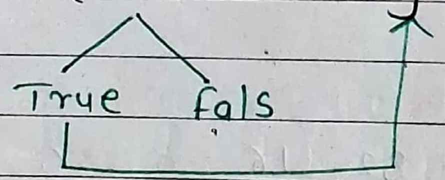
⑥ Conditional Operator :-

It is only Ternary operator in Java we can use with the help of syntax.

Syntax :-
operand 1 ? operand 2 : operand 3 ;



Example :-
System.out.Println(4>3 ? "4 is greater" : "3 is small");



→ S.o.P ("4 is greater");
OR

String s = 4 > 3 ? "4 is greater" : "3 is small";

Q) write a java program to check number is positive or negative.

```
Class Test {
 public static void main(String [] args) {
   int num = 10;
   system.out.println(num >0 ? "+ve" : "-ve";)
  }
}
```

Q write Java program to check a number is even or odd.

```
Class Test {
  public static void main(string [] args) {
    int num =10;
    String s = num % 2 = 0 ? "Even" : "odd";
    System.out.println(s);
  }
}
```

⑦ Assignment operator :-

we have only one assignment operator i.e =

Example :-
int a = 10;

Compound Assignment Operator
It is a combination of arithmetic & assignment operator.
(+= , -= , *= , /= , %=)

* Type Casting :-

Converting and datatype into another datatype is called as type casting.

Q. who can perform type casting?
① compiler
② programmer.

NOTE :-
① If the type casting is done by the compiler it is called as implicit type casting.

② If the type casting is done by the Programer it is called as explicit type casting.

we have two types of type casting based on datatype :

① Primitive Typecasting.
② Non-Primitive TypeCasting.

① Primitive Typecasting :-

- Converting one primitive datatype to another primitive datatype is called as primitive typecasting
- Primitive type casting cannot be perform on boolean
- we have two types of primitive typecasting.

① widening
② Narrowing .

① widening :-

- converting lower size datatype to higher sized datatype is called as widening.
- Both implicit & explicit typecasting can be perform in widening process.

Example :-

```
Class Widening {
 Public static void main(String [] args){
  int a=10;
```

```
double d = a; //widening done by Compiler
System.out.Println(d);   // o/p = 10.0
}
}
```

② Narrowing :-

- Converting higher sized datatype to lower sized datatype is called as Narrowing.
- Only explicit typecasting can be done in narrowing process.
- Narrowing can be done by using typecast operator.

Q. why narrowing is done explicitly?
→ In narrowing process there is chance of data loss and compiler will not take the risk so it is up to the programmer.

Example :-

```
class Narrowing {
 Public Static void main (String [] args) {
 double a = 3.14;
  int b = (int) a  // narrowing done by Programmer
 System.out.Println(b);
} }
```

① class widening {
```
class widening {
    public static void main (String[] args)
    {
        Char ch = '$';
        int b = ch;         // storing ASCII value
        system.out.println (b);   // 36
    }
}
```

② class narrowing {
```
class narrowing {
    public static void main (String [] args){
        int a = 64;       // 4
        char ch = 97;     // storing ASCII
                          Character to ch
        system.out.println (ch);  }  }
```

* Type Cast Operator :-

→ unary operator

syntax :

(datatype) var ;

* Non-Primitive Typecasting :-

- Converting one non-primitive datatype to another non-primitive datatype is called as non-primitive type casting.
- we know every class~~mate~~ name is non-primitive datatype.
- Therefore we can also say -

Converting one class type to another classtype is known as non-primitive typecasting.

* Methods :-

It is block of instruction which is used to perform some specific task.

Q. why we need method ?
To reduce the number of lines of code.

without method
```
{
    // 900 lines
}
```

with method
```
{
    // 300 lines
}
```

- **Syntax to declare/ Create method :-**

```
[modifier] returntype methodname ([formal argument])
{
    // Instruction
    [return [val/var/exp] ;]
}
}
```

Example : Main Method

modifier → Public static
returntype → void
method name → main
formal argument → (String[] args)

Public static void main (String[] args)

* method defination

method declaration: modifier returntype methodname([Formal arg])

method signature: methodname([Formal arg])

```
modifier returntype methodname([Formal arg])
{
    // instruction
    [return [val/var/exp] ;]
}
```

method body/ block/ implementation: the part from { to }

NOTE :-

1. Method can be created inside class block.

method Defination ⇒ method Declaration
+
method implementation

(2) we cannot created method inside a method nested method are not supported in Java

(3) we can create n number of methods in a class block.

* Modifiers :-

(1) modifier are use to modify / change after the behaviour of components like variable, method.

(2) we can access 2 type of modifier.
  (i) access modifier
  (ii) Non-access modifier.

1) Access modifier :-

(1) Acccess modifier use to change accessibility
(2) we have 4 access ~~Specifier~~ modifier
(3) also called access specifier.

(i) Public
(ii) protected
(iii) default
(iv) Private.

2) Non-access modifier :-
① it is used to change some behavior.
② multiple non-access modifie

(i) static
(ii) Non-static
(iii) final
(iv) synchronised
(v) native
(vi) transient.

* Returntype

① It is type of data return by the block of instruction
② Returntype is datatype i.e return by method.
③ if method will return the data it can only return primitive datatype or non primitive datatype.

Return type
- byte
- short
- int
- long
- float
- double
- char
- boolean

* If method will not return any data the return type will be void.

* Type of method based on Formal argument.

Based on formal argument:

- No-argument method
- parameterized method

method declared or created without formal argument is called as No argument method.

example :-

```
modifier returntype methodname ()
{
    // instruction
    return val/var/exp ;
}
```

* To call a method :-
we can call a method with the help of method signature.

example:-

```
public static void ~~main~~ add() (String[] args)
{
    int a = 100;
    int b = 100 ;
    S.o.p (a+b);
}  public static void main (string[] args)
        {
            add(); → method signature
        }
```

Sum call

Q> whgo calls Java main method.
→ JVM calls main method

* characteristics of method
- method get executed only when it is called.

example.
```
class ex {
 public static void display()
{
  S.o.p("display method");
}
Public static void main(String [] args)
{
  S.o.p ("Tanvi"). } }
```
↗ output

Q> what happen when method call ?
→

① The current execution of the method will be pose
② The Control will go to the called method
③ The execution of called method Start from begining
④ after execution of called method the control will get block to the caller.

(5) The execution of the caller resume.

NOTE :-
Caller method :- The method who calls another method is known as Caller method.

Called method :- The method who is getting called by another method is known as called method.

advantage :- easy to called

* Parameterized method :-

- Any method created / declared with formal arguments is called as parameterized method.

syntax :-

```
modifier returntype methodname (datatype var1, datatype var2.. )
{
    return (val/var/exp);
}
```

NOTE :-
The variable declared in the method declaration statement is called as formal argument.

Example :-

```
class A {
  public static void main(String [] args){
    add (100, 200) ;
  }            └→ actual argument
  public Static void int add (int a , int b)
  {                          ↳ formal argument.
    return a+b ;
  }
```

Syntax to call parameterized method :

method name (actual argument);

* Actual argument :-

These are the values or data given to the formal argument.

* Rules to call parameterized method :

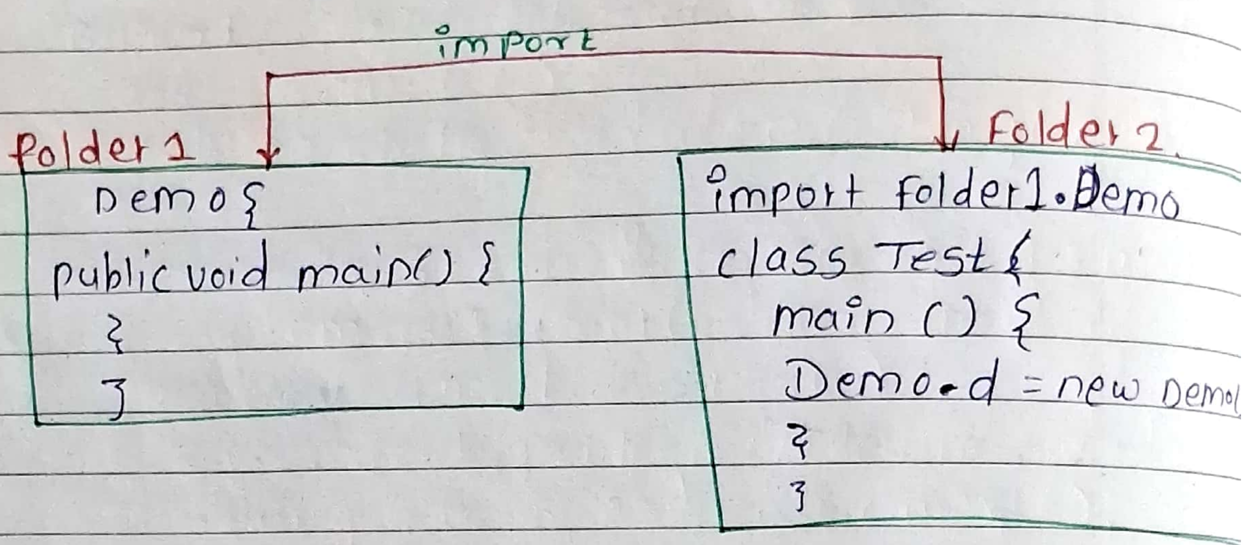
① The number of formal argument & actual argument should be Same.

② The type of corresponding formal & actual argument should also be same.

Example:-

```
Class demo {
public static void display(String s, int a)
{
  }
Public Static void main (string [] args)
{
  display ( 1, " Hi " );      → CTE
  display ( " Hi " );         → CTE
  display ( " Hi " );         → CTE
}
class Test {
public static void addofthree(int a, int b, int c)
                              local variable
{
  S.O.P (a+b+c);
}                    ↓ caller.
public static void main (string [] args)
{
  addofthree (10, 20, 30) - method calling statement
```

* Calling non-static methods of differen class in different Packages :-



* Dynamic Read :-

Reading the data when the application / program is under executing is the concept of dynamic Read.

we use Scanner class to implemen Dynamic Read.

① ~~Java~~ java.util.Scanner ;

① Scanner is a class
② It is present in Java.util Package.
③ It consist of non-static method which is use to take input from

user.

* Step to Read the data from the user by using Scanner :-

① Import java.util.Scanner.

② Create the object of Scanner class And initialize it with System.in

~~An~~

Scanner S = new Scanner(system.in);

③ We can read the data by calling non-static methods.

example :-

S.nextInt();

* Methods in Scanner class :-

Method signature	Returntype
nextByte();	byte
nextShort();	Short
nextInt();	int
nextFloat();	Float
nextDouble();	double
next();	boolean
nextline();	long
nextBoolean();	String
nextLong();	String

String will be Print till the Space ← next();

Program for ~~user~~ taking input from user.

```
import java.util.Scanner;
class Example{
  public static void main(String[] args)
  {
    Scanner S = new Scanner(System.in);
    System.out.print("Enter email : ");
    String email = S.next()
    System.out.print("Enter Password :");
    String passward = S.next();
  }
}
```

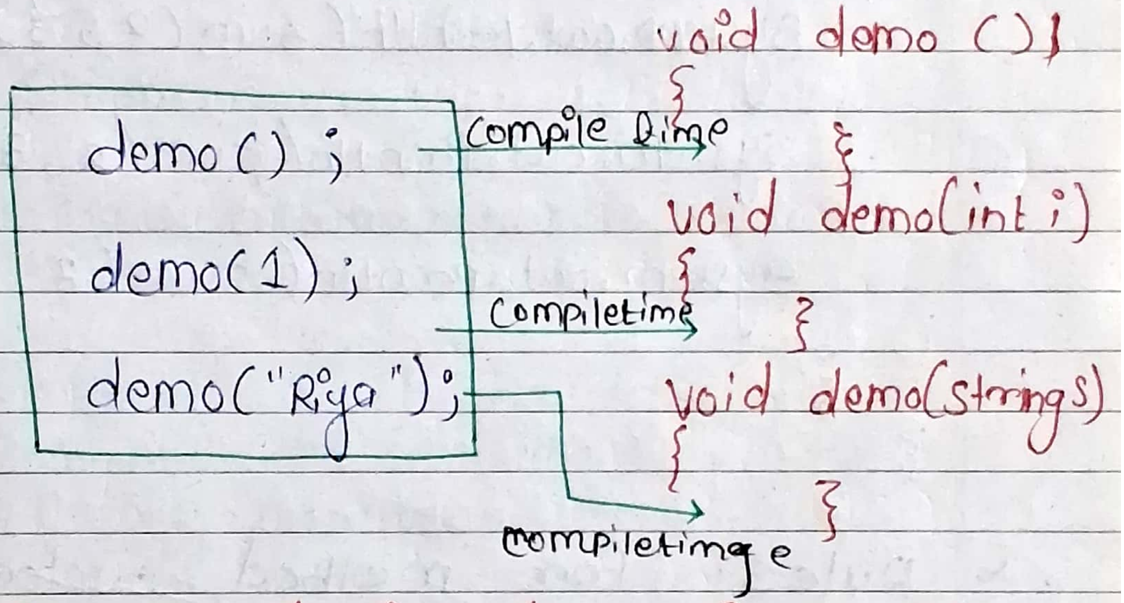
Q) Create a class APP1 take the user input of two number & prints their sum.

```
import java.util.Scanner;
class APP1 {
  public static void main(String [] args)
  {
    Scanner S = new Scanner(System.in);
    System.out.print(" Enter 1st No. ");
      int Num = S.nextInt();
    System.out.print(" Enter 2nd No.");
      Int Num2 = S.nextInt();
    Int Sum = Num + Num2;
    System.out.println(*Sum);
  }
}
```

* Method overloading :-

Methods declared with the same name but different formal arguments is called as method overloading.

Method overloading can be done only in the same class.



this method is done in compile time which called as compile time binding.

* Program for method overloading.

```
Package basic;
 public class overloading() {
   Public static void main ( int a, intb) {
                 int
```

```
        return a+b;
    }
    public static int sum(int a, int b, int c)
    {
        return a+b+c;
    }
    Public static int sum(int a, int b, int c, int d){
        return a+b+c+d;
    }
    public static void main(String [] args){
        System.out.Println(sum(2, 3));

        System.out.Println(sum(2, 3, 4, 6));

        System.out.println(sum(2, 8, 10));
    }
}
```

* Rules For method oveloading

1) Method name must be same

2) List of argument should be different either by type or by length.

Note:-
modifiers & returntype can be anything

True cond will be written in if block.
& false condⁿ will be written inside the else block

* Compile Time Binding :-
The process of binding the method call statements with the method body at ~~com~~ the compile time is called as compile time binding

* Decision Statement :-

- These are the statements which are use to implement the condition

- Decision statements are called as Conditional statements

- We have 4 Types of decision statements.
  1> if Statement
  2> if - else statement
  ~~3> if-else ladder~~
  3> else - if ladder
  4> Switch

① if statement :-
Syntax :-

```
if (condition) {
        True
         ↓
    // set instruction
}
```

example :- n = 10 ;

```
if (n>0) {
    S.O.Println("+ve numer");
}
```

NOTE :-

in if statement weather cond^n is true the instruction will execute

if the cond^n is false then NO instruction will execution

* If-else statement :-

① syntax :-

```
                 if (condition) {
True condition  → // instruction 1
                 }
                 else { instruction 2
false cond^n →   }
```

* Working of if-else statement :-

① In if-else statement if the condition is true then instruction 1 will execute. if it is false then ~~infr~~ instruction 2 will execute.

Q> Design a non static method which accepts age of person & return the eligibility for voting ?

```
Public class vote{
  Public static void main(String[] args)
{
    int age;
     vote obj = new vote();
    Classname               Classname
    Scanner
    S.o.Println( obj. eligible(19));
                       method name
 }
  public void String eligible (int age){
                      method name
    if (age >= 18) {
        S.O.Println(" eligible ");
    }
    else{
       S.O.P("Not eligible");
    }
 }
```

Q> Write a Java program to check a number is even or odd ?

```
public class Number {
   public static void main (String [] args)
   {
       int a=20;
       if (a%2==0){
        S.O.P(" Even ");
       }
```

ladder is used to climb the steps.


```
else {
        s.o.p ("odd");
    }
  }
}
```

* else- if ladder :-

Syntax :-

```
if (condition 1) {
      // instruction 1
}
else if (condition 2) {
      // Instruction 2
}
else if (Condition 3) {
      // Instruction 3
}
      :
      :
else {
      // instruction n
}
```

Q) write a program to find the greater number between 3 numbers

```
public class MaxThree{
  public static void main (String [] args)
  {
        int a=10;
```

```
int b = 30;
int c = 40;

if (a > b && a > c)
    S.O.Println (" a is greater");
else if (b > a && b > c) {
    S.O.Println (" b is greater");
else
    S.O.P ("c is greater");
}
}
```

Q> write a Java progra to check a value day & prints the corresponding day of the week. if the day is 1 it should be sunday & day 7 should be saturday & user enter some other value than 1 to 7 we should print invalid day

```
public class days {
    public static void main (String [] args)
    {
        int a = 3;
        if(a == 1)
            S.O.P ("sunday");
        else if(a == 2)
            S.O.P ("Monday");
        else if(a == 3)
            S.O.P ("Tuesday");
```

```
else if (a==4)
      S.o.P ("Wednesday");
else if (a==5)
      S.o.P ("Thursday");
else if (a==6)
      S.o.P (" Friday");
else if (a==7)
      S.o.P. (" Saturday");
else
      S.o.p. (" Invalid day");
```

Q) Write a Java program to check whether given number character is vowal or Consonant.

```
→ import Java.util.Scanner;
  class vowel {
    Public static void main (String[] args)
    { S.o.P (" Enter any character : ");
      Scanner s = new Scanner (System.in);
      char Ch = s.next().charAt(0);
      if (ch=='a' || ch=='e' || ch=='i' || Ch=='o' ||
            ch=='u') {
            S.o.P (" vowel");
      }
      else {
            S.o.P (" Consonant ");
      }
  }
```

* Switch Statement :

① Switch is a decision statement

② It is an alternative of else -if ladder statement.

Syntax :-

```
switch (val/var)
{
    case value/exp :
        {
            // Instruction 1
        }
    case value/exp :
        {
            // Instruction 2
        }
    .
    .
    .
    .
    default : {
            // instruction
        }
}
```

For switch value the allowed datatypes are char, integer, byte, short, ~~long,~~ String, enum.

Example :-

```
public class vowelorCons {
    public static void main(String[] args){
        char ch = 'b';
        switch (ch) {
        default : { S.o.p ("Consonant"); }
        break;
        case a : { S.o.p (" vowel "); }
        break;
        case 'e': { S.o.p ("vowel "); }
        break;
        case 'o': { S.o.p (" vowel "); }
        break;
        case 'u': { S.o.p (" vowel "); }
        break;
        case 'i': { S.o.p (" vowel "); }
        }
    }
}
```

② datatype which are not allowed for float, double, boolean, long.

③ in case value the allowed datatype are byte, short, int, char, strings, enum.

④ Datatypes not allowed case value are float, double, boolean & long.

interview question

Q) ~~Why~~ which way is more efficient either else if or switch ?

→

① switch statement more efficient because code readibility is increases and also the execution is faster in switch than else if ladder.

* looping statements :-

① This are the statements which are used to exeute the instruction multiple time

loop meaning :- Repeatation

② In Java we have 4 types of loops.

① While loop
② do-while loop
③ for loop
④ for-each loop (enhanced loop)

① while loop :

① while is a keyword

② syntax to use while loop :

```
While (condition) {
        True / False
     // Instruction
}  <- // updation statem
```

control will go out of the while block.

Example for while loop.

```
class whileloop {
  public static void main(String [] args){
      int a = 8 ;
      while ( a >= 5 ){
          S.O.pln( " clap " );
          a -- ;
      }
  }
```

~~Int~~ minimium iteration in while is 0

NOTE :- Iteration.

① The execution of instruction in a loop is called as iteration

Iteration for above while code :

iteration1	iteration2	iteration3	iteration4
a = 8	a = 7	a = 6	a = 5
8 >= 5	7 >= 5	6 >= 5	5 >= 5
True	True	True	True.
clap	clap	clap	clap
a = 7	a = 6	a = 5	a = 4

a = 4
4 >= 5
False

* Working of While loop :-

① In while loop first condⁿ will be check if it is True then only then instruction will exeute.

if it is false then instruction will not execute & control will go out of the block.

② Do-while loop :-

① Do & while both are keywoords

② Syntax

```
do {
    // instruction
    // updation statement
  }
while (condition);
```

Example for do while :-

```
int a = 0;
  do {
    s.o.pln(a);
  }
while(a>10);
```

infinite
zero

Q) Difference between while loop & do-while loop

While	Do-While
① first condition will be check then instruction will execute	first instruction will execute then the condition will ~~che~~ be check.
② minimium Namber of iteration is '0' (zero).	minimium Number of iteration is '1'
③ semicolon(;) isn't required	semicolon is mandatory.

* For loop :-

① For is a key coord

② It is the used loop

③ syntax :-

```
for( initialization①; condition②; updation④ ){
                          T/F
    // instruction③
}
```

(arrows: initialization → condition; condition T/F ↓ instruction; control goes to updation; updation → condition)

} ← Control go out of block.

Java doesn't provide actual address it provide modified address.

* ARRAY :-

Array is a contigous block of memory which is used to store multiple data's /values.

contigous → which has end
continaus → which doesn't have end

Q> Why we need an array?
→
we need an array to store multiple data under single name.

Q> How to create an array?
→
syntax for array declaration

datatype [ ] identifier = new datatype[size];

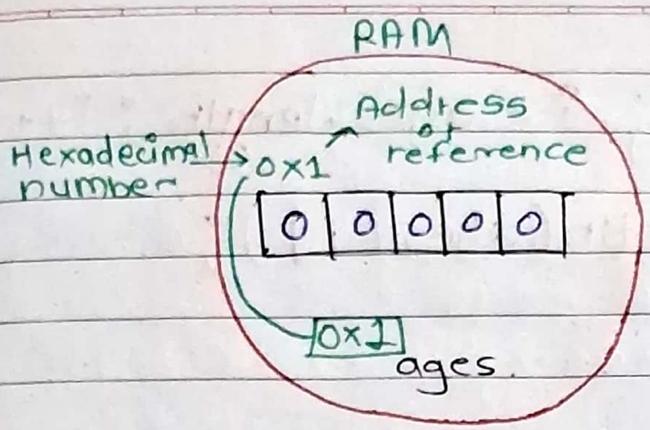
↑ subscript / array operator / square braces

Example :-

```
int [] ages = new int[5];
```

Fixed length memory allocation.

Size -1



* Characteristics of an array

① Only homogenous data are allowed

② The sized of the array is fixed on runtime.

③ Every array has two things.

i) Address
ii) Index

④ We can store the data or fetch data from an array by using address & index.

To ~~Print~~ Program

```
int [] ages = new int [5];
ages[0] = 10;
ages[1] = 20;
ages[2] = 30;
ages[3] = 40;
ages[4] = 50;
```

```
for (int i=0; i< ages.length; i++)
  {
     S.o.Println(ages[i]);
  }
}
}
```

NOTE :-
Every has length variable which is used to store the current length of the array. we can access the lengthe of the array by using identifier.length.

Case :- I

```
int arr[ ] = new int [-1] ;
```

we cannot create an array of negati size.
if we created we will gete
NegativeArraySizeException

Case - II :-

```
Int arr[] = new int [0];
```

we can create an array of size zero (0) if we try to store any element if that array we will get ArrayIndex

OutofBoundException

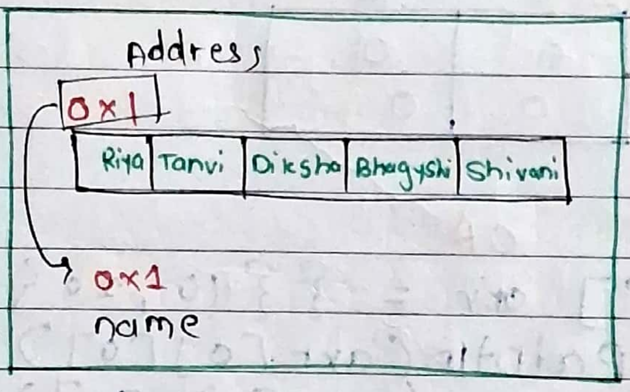
Case-3 :-

```
Int arr [] = new int[3];
  S.o.println(arr[4]);
```

if we try to access or fetch the index which is not inside the array we will get ArrayIndexOutOfBound Exception.

Q) How to declare & initialize the array in one line ?

```
Int [] ages= {10,20,30,40,50};
```



Program to Print 5names in one variable by using array

* Types of arrays :-
We have two types of array
1) 1-D array
2) Multi-dimensional array

* Multi-dimensional Array :-

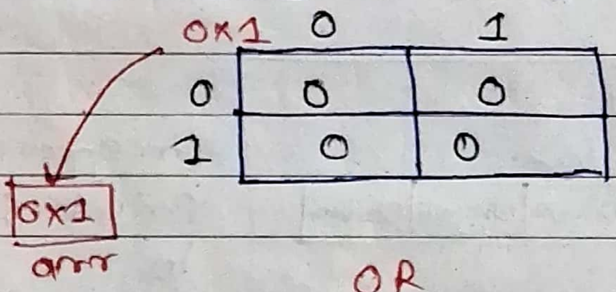
Multi dimensional has two types
i) 2-D Arrays
ii) 3D Arrays

i) 2D Array :-
Syntax :-

Datatype [][] identifier = new datatype [rows] [columns];
(subscript / array operator)

Example :-

```
int [][] arr = new int[2] [2];
```

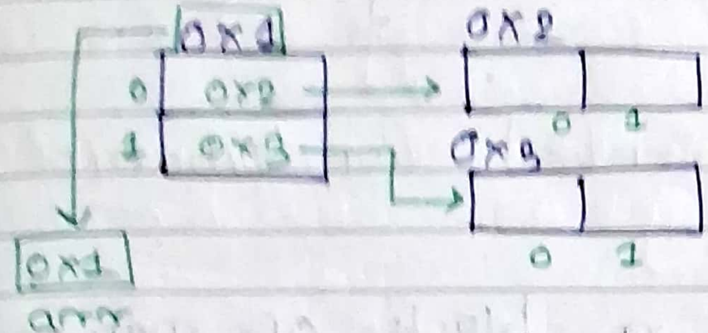


OR

```
{ int[][] arr = { {10, 20}, {30, 40} }
    S.O.Println(arr [0][0]);
    S.O.Println (arr [0] [1]);
    S.O.Println(arr [1] [0]);
    S.O.Println(arr [1] [1]);
  }
}
```

Output
10
20
30
40

int arr[][] = new int[2][2];



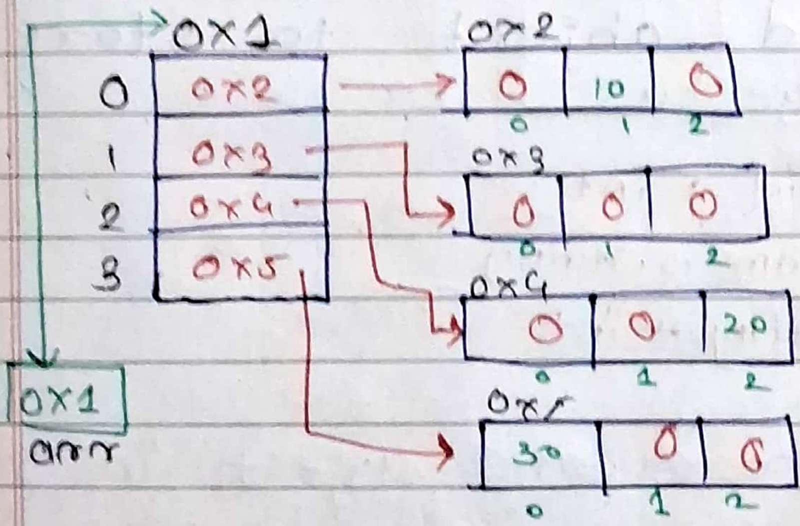
0x2[0] = 10
arr[0][0] = 10;

0x3[1] = 40;
arr[1][1] = 40;

To find the length of 0x1

0x1.legth
arr.length

for 4 rows + 3 columns



0x2[1]
arr[0][1] = 10

0x4[2]
arr[2][2] = 20

0x5[0]
arr[3][0] = 30;

```
for(int row=0;row<arr.length;row++)
{ for(int col=0;col<arr[row].length;col++)
  { S.o.print(arr[row][col]);
  } S.o.println();
}
```

col < arr[row].length
to know the length of 0 index row.

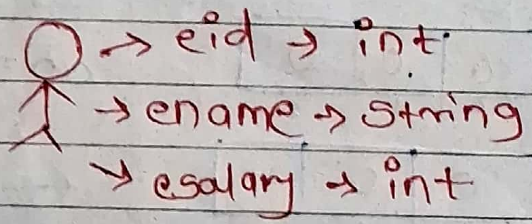
* Object oriented fundamentals

object :-
① It is a block of memory which can have multiple variables & methods present inside it.

② Every object will have address or reference.

Q Why we do need object ?

we need object to store heterogenous data



Steps to create object in Java

① Create a blueprint

if bluePrint already exist direct we can create an object

- object is a real world entity & it is block of memory
- used to store multiple heterogenous variable & methods.

if it doesnot exist.

② create an object by using syntax or new keyword.

Syntax :- new classname ();

Note :-
① ~~Ra~~ Blueprint specifies type of object
② IN Java, we create a blueprint by creating a class.

Example :-

```
class marker {
    public static void main (String [] args){
        s.o.println (new.marker());
        s.o.println (new. marker());
    }
```

output :- array.Marker@1d25189 } address

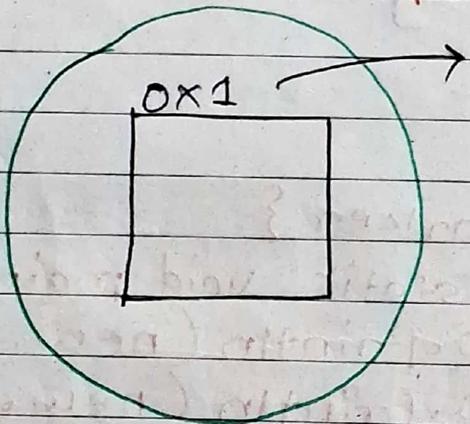
③ if we need to create a marker in programming, first we need to create a blueprint ~~from~~ for marker

== compare address

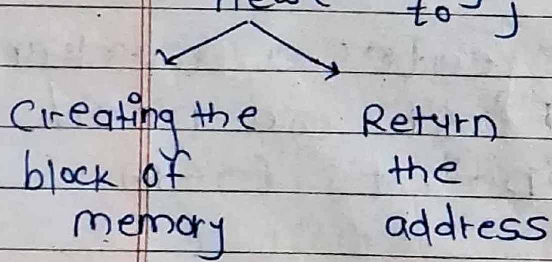
~~New :-~~

new :-

1. new is a keyword & operator
2. new create some block of memory inside the heap area
3. new will return the address of the block of memory (or reference of the same object)
4. either we can store the address in reference variable or we can print it.



new (mainly is used to)



CTS = compile time successfull

* Non-Primitive datatype :-

① In java every Classname is non-primitive datatype.

② Non-Primitive datatype can be used to create non-primitive variables

Reference variable :-
Variable declared or created to store NULL or ADDRESS of the same object is known as Reference variable

How to create reference variable
Syntax :-
Datatype identifier ;
ex -
Mobile m ;
Non-Primitive Datatype — reference variable.

Program

```
Package objectfundamental;

Public class Mobile {
    Public static void main(String[] args) {
        Mobile m = 10;          // CTE
        Mobile m1 = 'a';        // CTE
        Mobile m2 = "sheela";   // CTE
        Mobile m3 = null;       // CTSuccessful
// Storing the address of mobile in reference variable.
        mobile m4 = new mobile(); // CTS
    }
}
```

we can create multiple object.

Ctrl + Space

Ctrl + 1 .

* NOTE :-

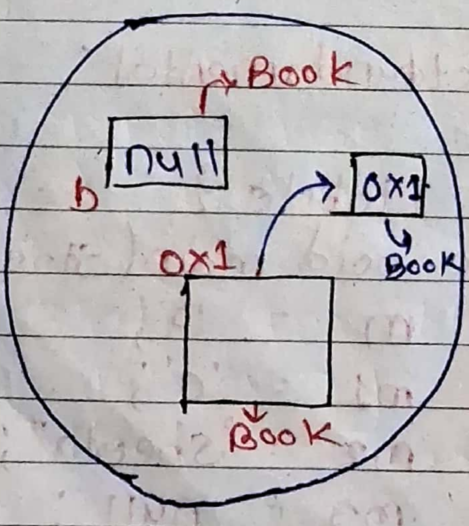
① We can create any number of objects & every object will have different address.

② We can create of the object of the class in the same class as well as in the different class.

③ We cannot store some different type of object in some different type of reference variable

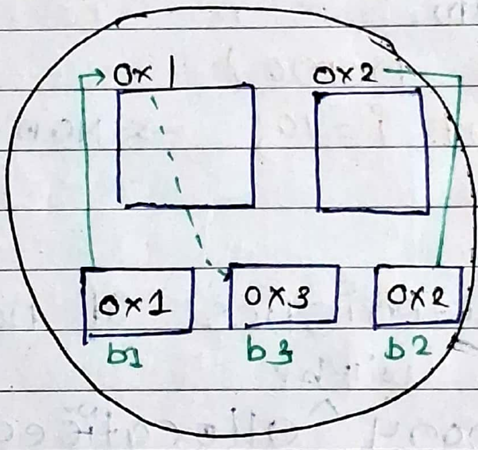
Program :-

```
class Book {
  main () {
    Book b = null;
    Boob b1 = new. Book();
  }
}
```



* Copy the reference of multiple object.

```
Package objectfundamental;
class Book {
public Static void main(String[] args){
    Book b1 = new Book();
    Book b2 = new Book();
    S.O.Println(b1 == b2);   // false
    Book b3 = b1;
    S.O.Println(b3 == b1);   // true
    S.O.Println(b2 == b3);   // false
}
```



NOTE:-
An object can be refered by multiple variables.

~~Non-Stal~~

Non-static

① Non-static variable :-

Any variable created or declared inside class block without prefix with static keyword is known non-static variable.

```
class Demo {
    int i;        → Non-static variable
}
```

Program.

```
Class Demo {
    int i=10;     → Non-static variable.
}
```

Characteristics of non-static variable :-

① Memory will be allocated inside the object.

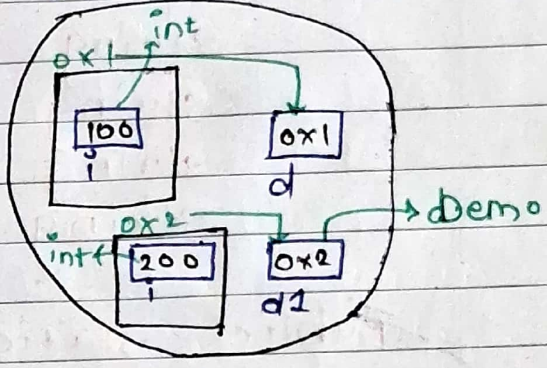
② We can not use non-static variable without creating the object.

③ Non-static variable will be allocated once in every object created for that class.

- it is a non-access modifier
- The component without static keyword like variables, method, initializer / block.
- Any variable created or declared inside the class block without static keyword.

④ Non-static variable are initialize with their default values.

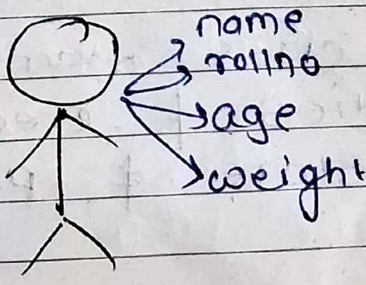
⑤ We can use non-static variable only with the help of object reference or reference variable (object Address)



Program

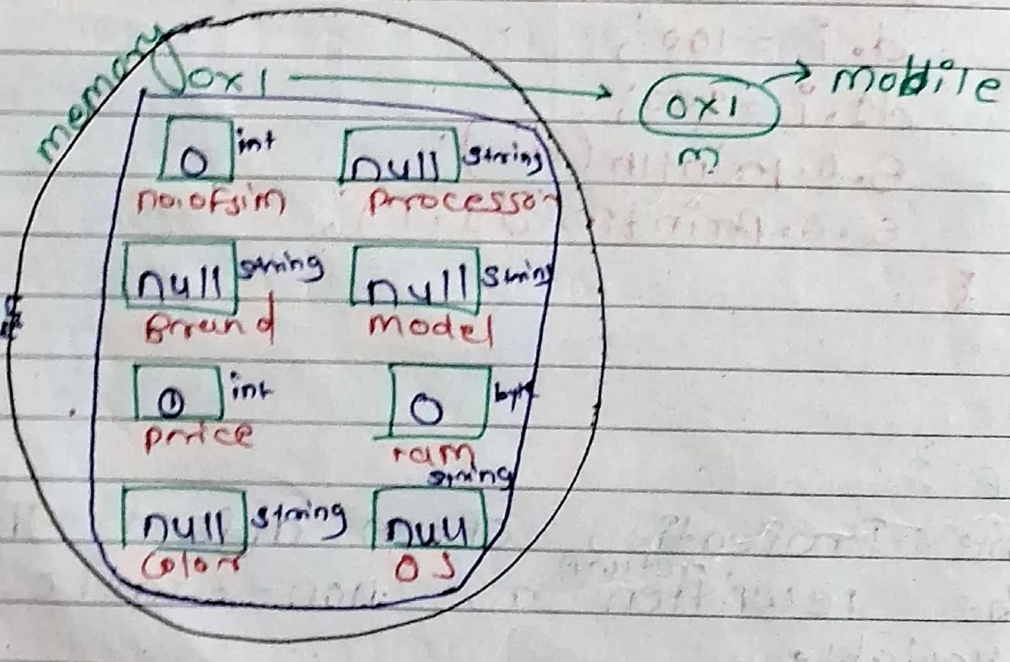
```
class Demo {
  int i;
  main() {
    Demo d = new Demo();
    Demo d1 = new Demo();
    d.i = 100;
    d1.i = 200;
    S.o.println(d.i);      // 100
    S.o.println(d1.i);     // 200
  }
}
```

NOTE :
The Properties of an object should be return as non-static variable



```
public class mobile {
    intt noofsim;
    String processor;
    int Price;
    byte ram;
    string brand;
    string model;
    string color;
    String os;

public static void main(String[] args){
    Mobile m = new Mobile ();
    }
}
```



Q) Create a class ~~Stacr~~ car figure out 5 properties & create 5 object of car store & print their properties.

12 + 10 + 10 + 11 - 9 + 8 .


```
int a = 12;

        12      10    10   11         9       8
int b = a-- + a-- - a + a++ + a-- - --a + --a;
S.O.P(a);          11  12
S.O.P(b);          11  42
```

* Constructor :- [To load NSV & NSM inside the object / To initialize NSV inside the object]

① It is a special non-static method method or non-static member of the classs, which does not have return type

② It is used to load & initialize the non-static members inside object.

Syntax :-

```
[Access modifier] classname([formal argument])
{
}
```

Characteristics of Constructors :-

① The constructor name should be same as classname

② if class has any modifier same modifier should be applicable for constructor.

non-static member includes

non-static variable | non-static method.

③ Constructors doesnot have returntype because inside constructor we write initialization code.

④

case 1:

Before compiling
```
class Marker{

}
```

After compiling
```
class marker{
    Marker(){      // compiler

    }
}
```

case 2:

Before compiling
```
class Demo {
    Demo () {
        S.O.P("Demo Con");
    }
}
```

After compiling
```
class Demo{
    Demo (){
        S.O.P("Demo Constru");
    }
}
```

~~Example for~~
Types of constructor.

i) Default Constructor
[Predefined constructor]

ii) user defined constructor.

NOTE :-
if programmer fetch to create a constructor then compiler will add default constructor in the class.

Default constructor is nothing but No-argument constructor.

Example :-

default No-argument constract

```
class Marker () {
Marker()
  {
     S.o.P (" Marker constructor");
  }
}
```

① User-defined Constructor :-
The constructors which are created by Programmers iso called as user defined Construct

Types of user defined constructor

i) No-argument constructor

ii) Parameterized Constructor.

i) No-argument constructor :-

A) Any constructor created without formal argument is known as no-argument constructor.

when we use 'new' then the memory is created inside the heap area for object.

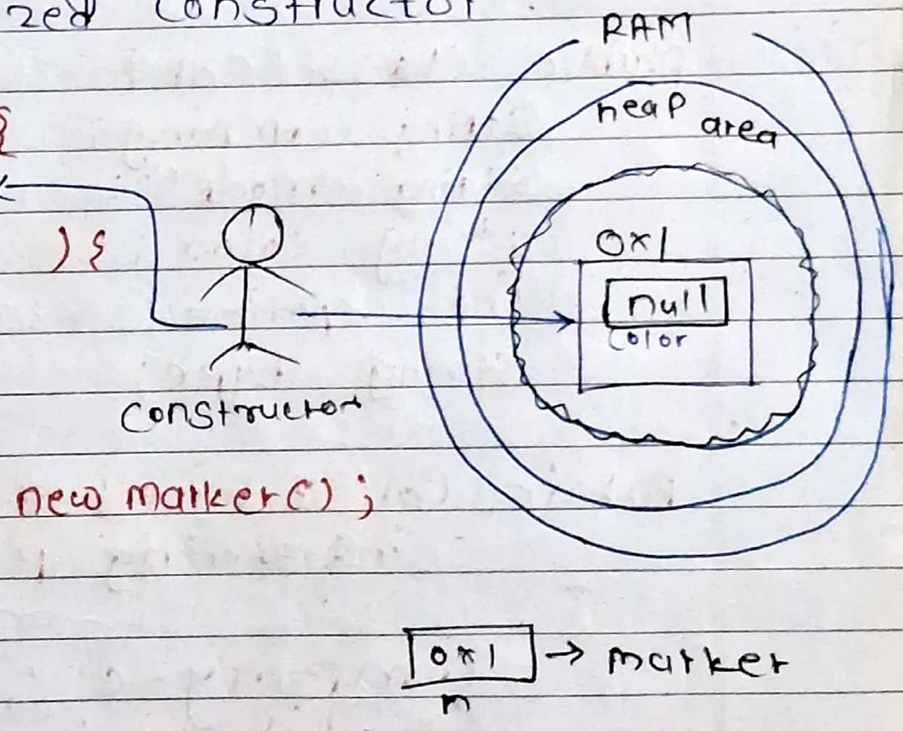
```
Public class Marker {
  public Marker(){
      S.o.println(" Marker Constructor");
  }
 }
 public static void main(String [] args){
     Marker m = new Marker();
     Marker m1 = new Marker();
 };
}
```

ii) Parameterized Constructor :-
Any constructor created with formal arguments is called as Parameterized constructor.

program:

```
class Marker{
  String color;
  Marker(          ){

  }
  main(   ) {
   Marker m = new marker();
  }
```



```
public class Marker{
    String color;
    public Marker (String a)
    {
        color = a;
    }
    public static void main(String [] args
    }
    Marker m = new marker("Black");
    Marker m1 = new Marker("Blue");
    S.o.p (m.color);        // Black
    s.o.p (m1.color);       //Blue
  }
}
```

```
ptublic Class Car {
    String company;
    String model;
    string color;
    int price;
    String type;

Public Car(String c; String m; String col,
        int String p; String type)
  {
      company = c;
      model = m;
      color = col;
      Price = P;
      type = t;
  }
```

```
public static void main(String [] args)
{
  Car c = new Car("BMW", "x++", "black",
           1900000, "Automatic");
   s.o.pln("company :- " + c.company);
   s.o.pln("model" + c.model);
   s.o.pln("color" + c.color);
   s.o.pln("Price" + c.Price);
   s.o.pln("type" + c.type);
}
```

* This keyword :-

① This keyword is used to hold current object address

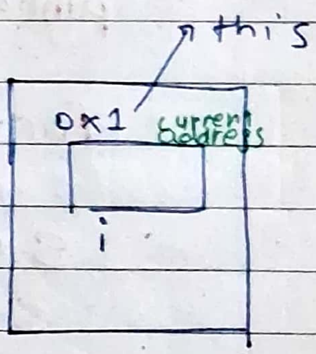
② It is a keyword must be written in lowercase letters.

Example :-

```
class Demo {
   int i  → N.S.V
   Demo (int i)
              Local variable
   {
      this.i = i;
   }
} main() {
 Demo d = new Demo(2)
```

In `this.i = i;` — `this.i` is the non static variable, `i` is the local variable.

In `new Demo(2)` — constructor calling statement



* Constructor Overloading :-

- Creating the multiple constructor with different formal argument within the same is known as Constructor overloading.

Advantage :
it gives multiple options to the programmer.

Program :-

```
public Employee{
    String Eid ;     } Non static variable
    String   name ; }
                         local variable
                             ↓
  Public Employee( String Eid ) {
      this .Eid = Eid } ;
  } non  ↑      ↑_____ local variable
   static variable
  public Employee( String Eid , String name)
  { this . Eid  = Eid ;
      this . name = name ;
  } // storing  a current address

  main( string [] args) {
    Employee E = new Employee( "ABC@123", "John");
  Employee E1 = new Employee("XYZ@123 ", "JOE");
constructor
calling     } S.o.P(E.id + "      " + E1.id + "      "+E1.name);
statement }
  }
```

blueprint is a class.

chaining: one hook is tie with another hook

* Constructor Chaining:-

① Calling one Constructor from another Constructor is called Constructor chaining

② Constructor chaining can be perform in the same class

③ we can call one constructor from ~~athore~~ another Constructor by using this (); (this called) statement.

Advantages:-
- Reduce the number of lines of code
- DRY (Donot Reapeat Yourself).

Program:-

```
class Employee{
  ~~Employee~~
    int eid;
    string ename;
    double sal;
  Employee(int eid){
    this.eid= eid;
  }
  Employee(int eid, String ename)
  {
    this(eid); this calling statement.
    this.ename = ename; }
```

Here we are called the above constructor for eid.

```
Employee (int eid , String ename , double sal)
  {
here we called above constructor → this ( eid , ename );
    this.sal = sal ;
  }
  main (String [] args) {
   Employee E = new Employee ( 1 , "Riya");

   S.o.println(E.eid + " " + E.ename + " " + E.sal );
  }
}
```

Q) Create a class Bank with Properties account_number , IFSC_code & ~~Pass~~ Bank-name. Create 2 Bank account & store it & Print their details By creating multiple Constructor.

```
class Bank {
  Long acc_no ;
  Long IFSC;
  String Bname;
  Bank( Long acc_no)
    {
       this. acc_no = acc_no ;
    }
  Bank( Long acc_no , Long IFSS )
    {
       this( acc_no );
       this. IFSC = IFSC ;
    }
```

```
Bank ( Long acc_no , Long IFSC , String Bname)
{
    this(acc_no , IFSC );
    this. Bname = Bname;
}
main ( String [] args)
{
    Bank B = new Bank (" 98930", 77898,
                        " SBI ");
    Bank B1= new Bank (99888 , 97789,
                        " BOB");
    S.o.Println (B. acc_no+" "+B.IFSC+" "+
                        B. Bname)
    s.o.p(B1.acc_no+" "+B1.IFSC+" "+B.Bname);
}
}
```

* Non Static Method :-

Any method declared or Created without static keyword is known as non-static method.

example:-

```
public void test() {

}
```

characteristics of non static method

① The memory for non-static method

will be allocated inside the object.

② We cannot call a non-static method without creating an object

③ A non-static method will be allocated in every object created for that class

④ Non-static ~~context~~ method is also known as Non-static context.

Non-static context :-
A block which belong non-static method is known as non-static context.

*** NOTE ***
① Inside any non-static context we can use both static & non-static members directly.

② ~~The~~ we use non-static method to access non-~~sat~~ static variable.

* Static :-

① static is a keyword.

② static is a modifier

③ With the help of static keyword we can create static members in a class.

④ we have 3 static members
   i> static variable.
   ii> static method.
   iii> static block / initializer

i> static variable :-
Any variable created inside the $ class block with static keyword is known as static keyword.

Syntax to create static keyword

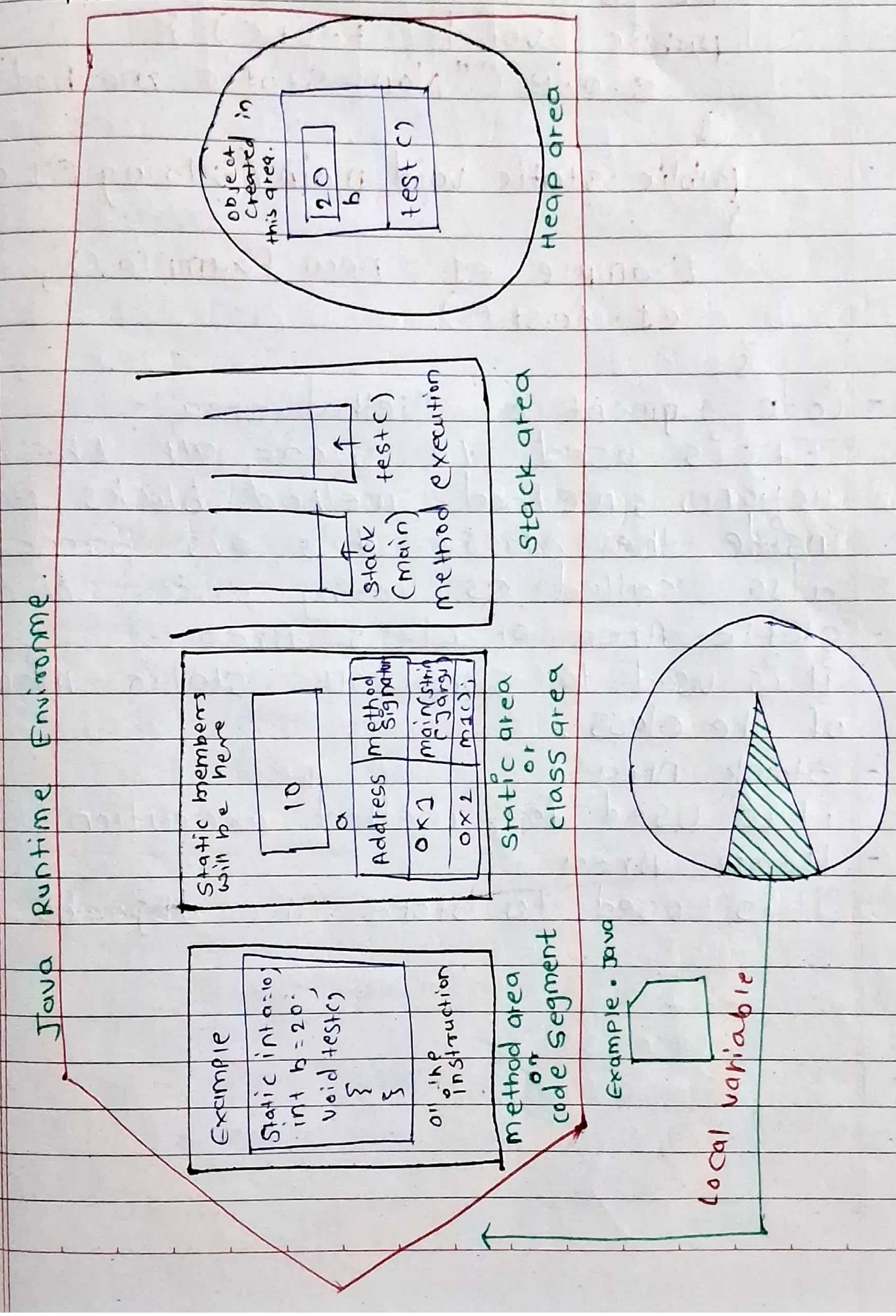
static datatype identifier;

Example :-

```
class Demo {
    static int a = 10;
}
```

Java Runtime Environment :-

Memory Description static & Non Static Member.



```
public class Example{
  static int a=10;
   int b = 20;
   public void test() {
      s.o.p("Non-Static method");
   }
  public static void main(String [] args)
  {
     Example e1 = new Example();
     e1.test()
  }
```

- Code segment or Method area :-
  It is used to store all the instructions, method, method blocks present inside the class this is process is also called as loading Process of class
- Static Area or class Area :-
  it is used to store all static members of the class.
- Stack Area
  it is used for method execution.
- Heap Area
  it is used to store the objects.

Object Oriented Programming

* Oops :

* static Initializer :- (static block) :-
A block prefix with static keyword is called as static initializer or static block.

Syntax :-
```
static {

}
```

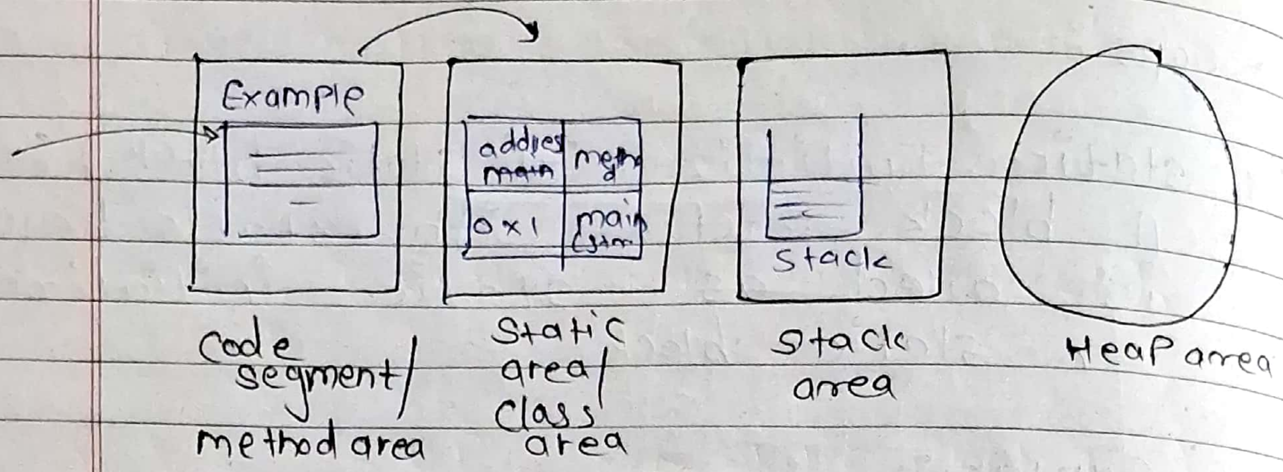
static blocks will execute during loading process of a class.

if we have multiple static block then execution will be done in top to bottom order.

```
Public class Example {
  static {
    S.o.p (" Static block ");
  }
  public static void main (String [] args)
  {
    S.o.p (" main start" );
    S.o.p (" main end");
  }
}
```

output
// Static block
// main start
// main end.

JRE




```
public class Example {
    static {
        s.o.p ("SB");
    }
    static {
        s.o.p("SB1");
    }
    public static void main(String [] args){
        s.o.p ("Main Start");
        Example e = new Example();
        s.o.p ("Main End");
    }
}
```

output

SB
SB1
Main Start
Main End

2) Non-static Initializer / Non-static block :-

Any block created static keyword is known as non-static initializer or block.

Syntax :-

```
{

}
```

Characteristic for Non-static block :-

① Non-static block will execute when the object is created.

② Non-static block will execute once for every object creation.

③ ~~Non-Sta~~ if we have multiple non-static block the order of execution will be top to bottom order.

```
Public class Example {
  {
      S.o.p(" Non-static block ");
  }
Public static void main(String [] args) {
    Example e = new Example();
  }
}
```

output

For multiple Non static block.

```
Public class Ex {
    {
        S.o.p("Non-static Block");
    }
    {
        S.o.p("Non-static Block 1");
    }
    Public static void main (String [] args)
    {
        Ex e = new Ex();
        Ex e1 = new Ex();
    }
}
```

output.

Non-static Block
Non-Static Block 1
Non-Static Block
Non-Static Block1

Q> program to know order of execution

```
Public class Demo {
    static {
    ① S.o.p ("Static Block");
    }
    {
    ③ S.o.p("Non-static Block");
    }
    Demo() {
```

Because of new keyword Non-static Block will execute before constructor Block.

① new — we are creating object.
Demo() ② — constructor

```
④ S.o.P("Demo Constructor");
  }
public static void main(String [] args)
  {
  ② S.o.P(" Main Start ");
     Demo d = new Demo();
  ③ S.o.P(" Main End "));
  }
}
```

Output

Static Block
Main start
Non-static Block
Demo constructor
Main End.

classmate
Date
Page

Object Oriented Programming :-

It is a Paradigm / design which help a programmer to built real world application like amazon, flipkark, etc.

It is a bottom - up approach for program.

Q) How to implement Oops ?

① Analyze the real world object
   i) Identify its properties
   ii) Identify its actions / behaviours

② Create a blueprint

③ Instantiate the class (creating the object)

NOTE :-
Bottom - up approach is that we don't create the object directly.
First we analyze the object & study about its properties & behaviours, then creating the blueprint & then creating the object.

We have 4 design Principles

1. Encapsulation
2. Inheritance
3. Polymorphism.
4. Abstraction

1) Encapsulation:-

The process of binding properties & behaviours of an object into a single unit is known as encapsulation

How to achieve Encapsulation
we can achieve Encapsulation by using class

defination
The wrapping up of properties & action of an object together is known as encapsulation

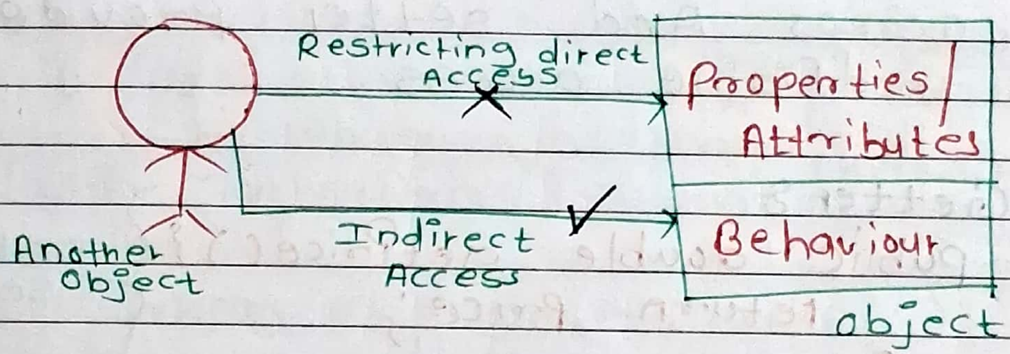
ATM
Properties / Attributes { bankname, location, Amount

Action / behaviour { withdraw() deposit() balance enquiry()

Advantages of Encapsulation :-

① Data hiding
② Data security.

Data Hiding :-
Restricting the direct access and Providing the indirect access ~~through~~ to the Properties of an object is called as data hiding.



Q How to achieve data hiding?
We can achieve data hiding with Private keyword. (Access modifier)

NOTE :-
By making all properties as Private.

Program

```
class Product {
  Private double price;
}
class customer {
  main () {
```

we are ...
the constructor

Date
Page

```
Product P = new Product();
P.Price = 2000;   → CTE
}
}
```

To achieve indirect access
We can achieve indirect access with the help of getters & setters.

Where getter provide readable access And setter provide modifiable access.

Getter:

```
public double getPrice(){
    return price;
}
```

} Data security

Setter :-

```
Public void setPrice(double price)
{
    this.price = Price;
}
```

} Data validation

Example:

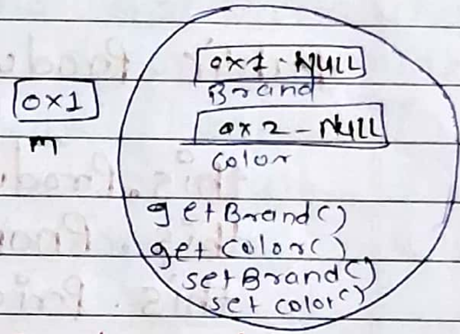
```
class Marker {
    private String Brand;
    private String Color;
```

getter → Redable Access

setter → Modifiable Access.

```
public String getBrand() {
        return Brand;
    }
public void setBrand (String Brand)
  {
        this.Brand = Brand;
  }
public String getColor() {
        return color;
  }
public void setColor(String color) {
    this.color = color;
  }
}

class MarkerDrive {
  main() {
   Marker m = new Marker();
   m.setBrand ("camlin");
  }
}
```



Question :-

Q. What is an Encapsulation?

Q. How to achieve Encapsulation?

Q. Advantages

Q. What is Data hiding

Q. How to ~~data~~ achieve data hiding

Q. How to achieve data security.

Date
Page

Q) Create class Product with properties Product_id, Product_Name & Price & Encapsulate the class & stored 5 products & Print their details.

```
Public class Product{
    Private int Product_Id;
    Private string Pname;
    Private double Price;

  Public Product (int Product_Id, String Pname,
                  double Price){
    this.ProductId = ProductId;
    this.Pname = Pname;
    this.Price = Price;
  }

  Public void display(){
    S.O.P("Product ID :-" + Product_Id);
    S.O.P("Product Name:-" + Pname);
    S.O.P("Product Price :-" + Price);
  }

  Public static void main (string[] args)
  {
    Product P1 = new Product(101, "Laptop", 50000);
    Product P2 = new Product(102, "Mobile", 20000);
    Product P3 = new Product (103, "Tablet", 300000);
  Product P4 = new Product (104, "Headphone", 5000);
  product P5 = new Product(105, "Smartwatch", 10000);
```

```
P1.display();
P2.display();
P3.display();
P4.display();
P5.display();
}
}
```

Q) What is Encapsulation?

Encapsulation means binding up Properties & behaviour of an object into a single unit.

How to achieve Encapsulation?
By using class.

Advantages
① Data hiding
② Data security

What is data hiding?
Restriction in direct access and Providing indirect access to the another object.

How to achiev data hiding?
By using Private keyword

How to achieve data security?
With the help of getter & setter method.

* Relationships in Java :-

The connection between two objects is called as Relationship

we have 2 type of relationship in java

i> HAS - A Relationship

ii> Is - A Relationship

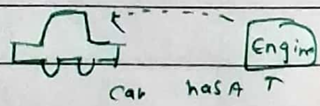
i> HAS-A Relationship :-

When the one object is dependen on another object such relationship is known as HAS - A Relationship.

OR

When the object is present inside ~~object~~ another Object is known HAS-A Relationship.

Examples of HAS-A Relationship

① Car has a Engine 

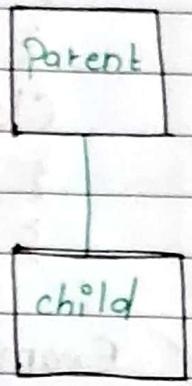
② Marker has a Ink

③ Laptop has a Ram

④ Phone has a simcards

ii) IS - A Relationship :
Any parent & child relationship is known as IS - A Relationship

Example :-
Car IS A vehicle
Camline is a Marker
Shoe IS A Marker
Mango IS A fruit



Q) What is the advantage of IS-A Relationship
We achieve Inheritance with a IS-A Relationship.

Inheritance :-

The process acquiring all the members of one class to the another class.

How to achieve inheritance in Java?
we can achieve inheritance by using two keywords

i) extends 2 classes & 2 interface
ii) implements. 1 class & 2 interface

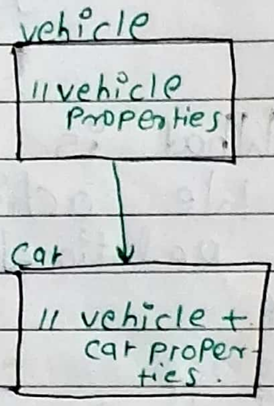
(i) extends :- we perform extends when Bet" 2 classes & 2 interfaces

Syntax :-

```
class ParentClassname
 {
 }
 Class childclassname extends ParentClassname
 {
 }
```

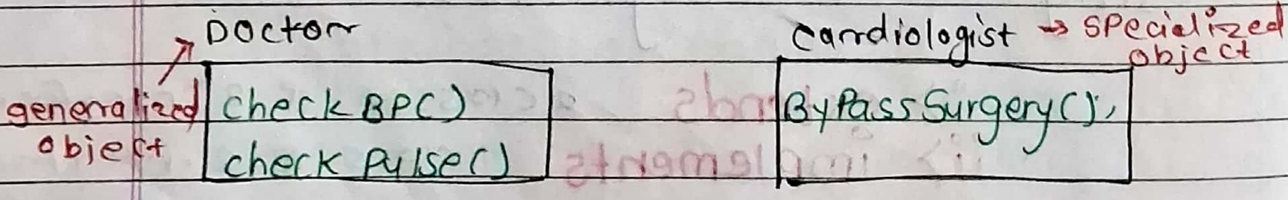
Example :-

```
Class vehicle {
}
class Car extends vehicle
{
}
```



Example 2 :-

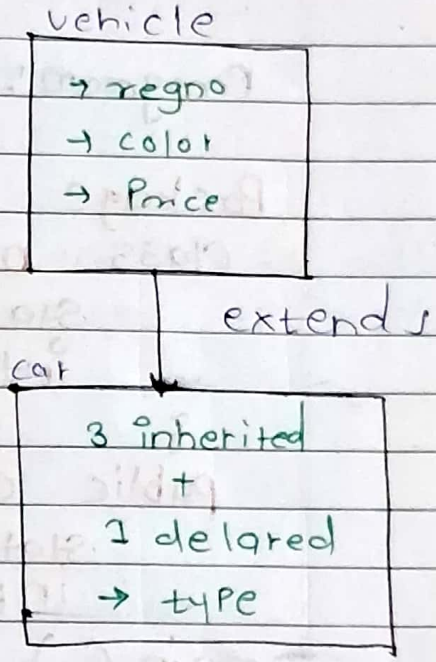
```
Class Doctor {
}
class Cardiatogist extends Doctor
{
}
```



We can inherite the class into another class in same Package

Example 3 :-

```
class vehicle {
    String regno;
    String color;
    double Price;
}
class car extends vehicle
{
    (3)
     +
    String type;
}
```



All the Properties of vehicle Present inside the Car class.

Inheritance With respect to static & non-static Members

	Static member		Non-static Memb.	
	Parent	child	Parent	Child
Parent class	✓	✗	✗	✗
child class	✓	✓	✗	✗
parent object	✓	✗	✓	✗
child object	✓	✓	✓	✓

Ways of accessing

Program:-

Package inheritance; // both classes present in same package.

```
class Demo {
    static int a=20;
    int b=20;
}
public class Test extends Demo {
    static int c=100;
    int d=200;
main () {
    S.o.p( Demo.a);
    S.o.p( Demo.b);   // CTE
    S.o.p( Demo.c);   // CTE
    S.o.p( Demo.d);   // CTE
    // with Parent classname we can
    // access only Parent
    // static member
}
* // with child classname we can access
  // both Parent & child static member.
    S.o.p(Test.a);
    S.o.p( Test.b);   // CTE
    S.o.p( Test.c);
    S.o.p( Test.d);   // CTE.

// with Parent object we can access both
// static & non static member of only
// parent
Demo obj = new Demo();
S.o.p(obj.a); ~~// CTE~~
```

```
S.o.P(Demo.b);
S.o.P(Demo.c);   // CTE
S.o.P(Demo.d);   // CTE.

// with child object we can access static
// & non static both members of Parent
// & child

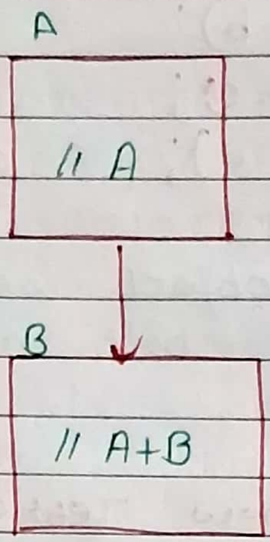
Test t = new Test();
S.o.P(t.a);
S.o.P(t.b);
S.o.P(t.c);
S.o.P(t.d);
}
}
```

* Type of Inheritance

In Java we have 5 types of inheritance

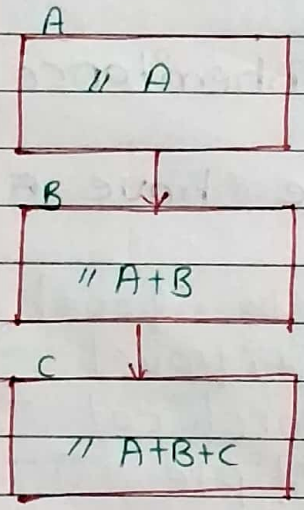
1> Single Level Inheritance
2> MultiLevel Inheritance
3> Hierarchical Inheritance
4> Multiple Inheritance
5> Hybrid Inheritance

1>. Single Level Inheritance :-
When the level of inheritance is one such type of inheritance is known as single level inheritance.



2> Multilevel Inheritance :-

When the level of inheritance is more than one such type inheritances are called as multilevel inheritance.



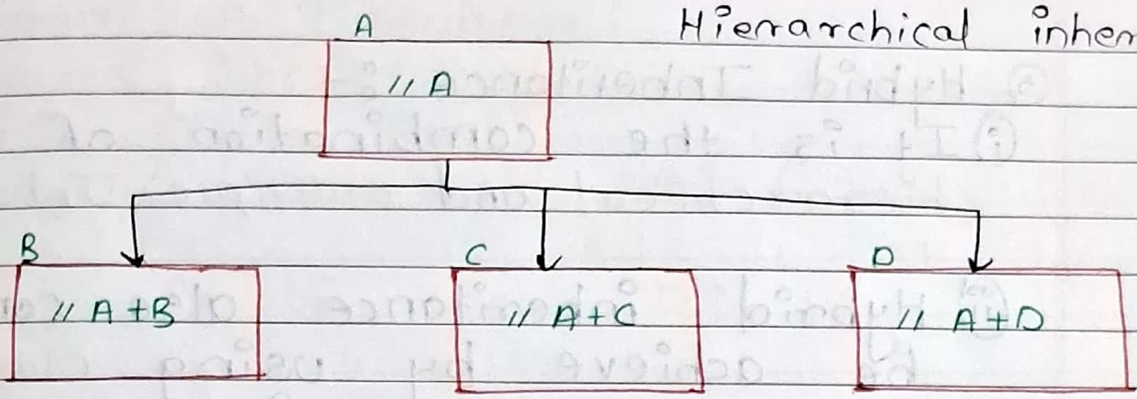
3> Hierarchical Inheritance :-

~~tA~~ When one Parent class has more than one child at the same level such type of inheritance are called as hierarchical inheritance.

super() → statement should be the first statement

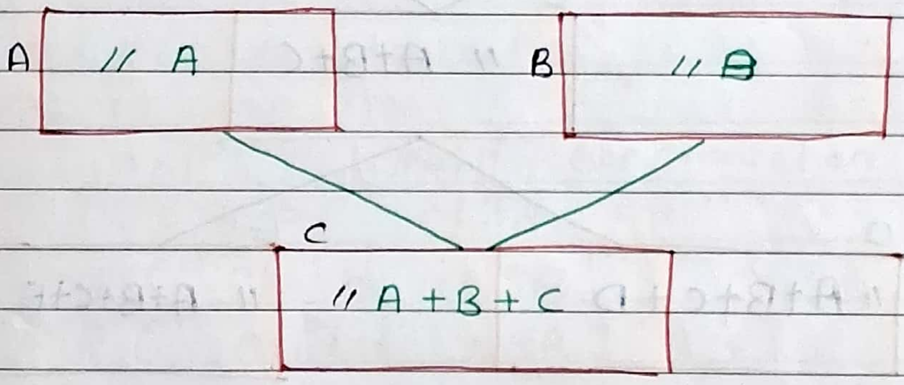
return type → should be the last statement

Hierarchical inheritance



* 4> Multiple Inheritance * *

(i) When one child has more than one Parent such type of inheritance are called as Multiple inheritance.



(ii) Multiple inheritance cannot be achieve by using classes.

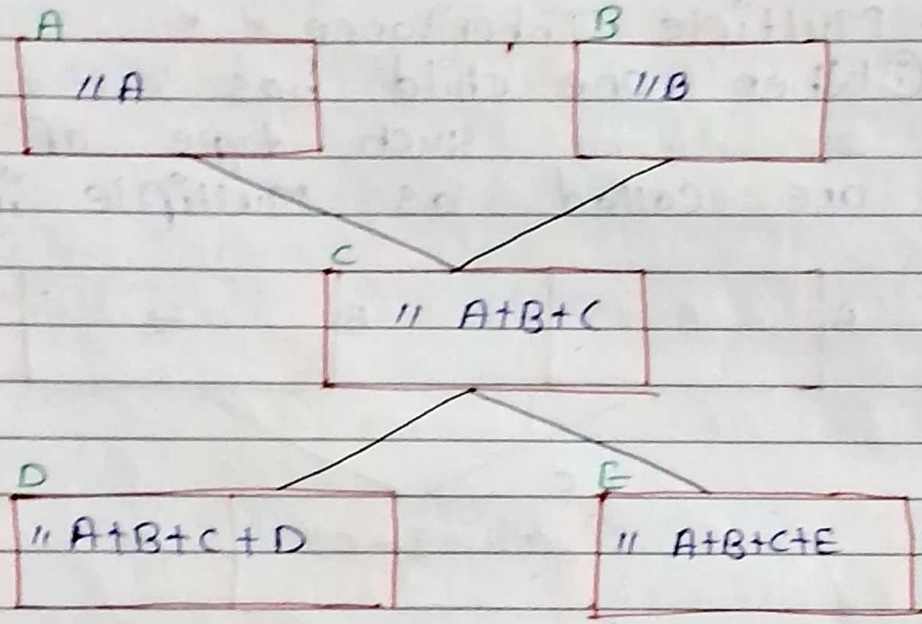
(iii) We can achieve multiple inheritance by using interface.

(iv) If we try to achieve multiple inheritance by using class we will get diamond Problem.

⑤ Hybrid Inheritance :-

(i) It is the combination of hierarchical and Multiple Inheritance

(ii) Hybrid inheritance also cannot be achieve by using class because multiple inheritance is involved between



classmate
Date
Page

* super ( ) statement :-

① Super call statement is used to call Parent calass constructor

② if Programmer fails to write super call statement then compiler will add super ( ) statement & it will call no argument constructor of parent class.

Rules of super() statement :-

① super() statement is should be the first statement first in the constructor body

② super() & this() cannot be written together.

③ Program :-

```
Package Inheritance;
public class Person{
    String name;
    int age;
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

```
class Student extends Person {
    int rollno;
    student (String name, int age, int rollno)
    {
        super (name, age);
        this.rollno = rollno;
    }
}

Package inheritance;

class StudentDriver {
  public static void main (String [] args) {
    Strudent s = new student ("Riya", 18, 41);
    s.o.p( s.name + " " + s.age + " " + s.rollno);
  }
}
```

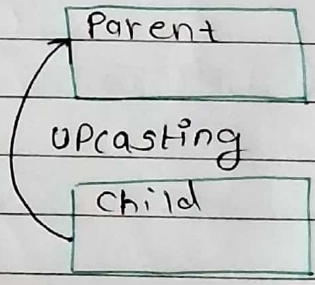
upcasting need Inheritance
+ for Downcasting needs
upcasting.

* Non - Primitive Typecasting :-

Converting non-primitive datatype ito another non - primitive datatype is known as non - primitive Typecasting
OR
Converting one classtype to another classtype is known as non - primitive typecasting.

we have 2 type non - primitive typecasting:
i> Upcasting
ii> Downcasting

i> upcasting :-
Converting child type reference to parent type is known as upcasting



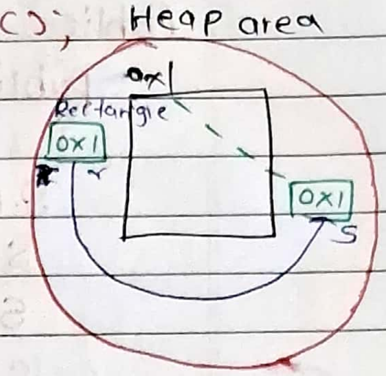
Parent obj = new child ();

example:-
```
class Rectangle extends Shape
{
}
class Shape {
}
class Driver{
  main() {
```

upcasting done by the both
programmer & compiler.

control + shift + O

Rectangle r = new Rectangle();
shape s = r; // compiler
grammer ← // shape s = (shape) r;

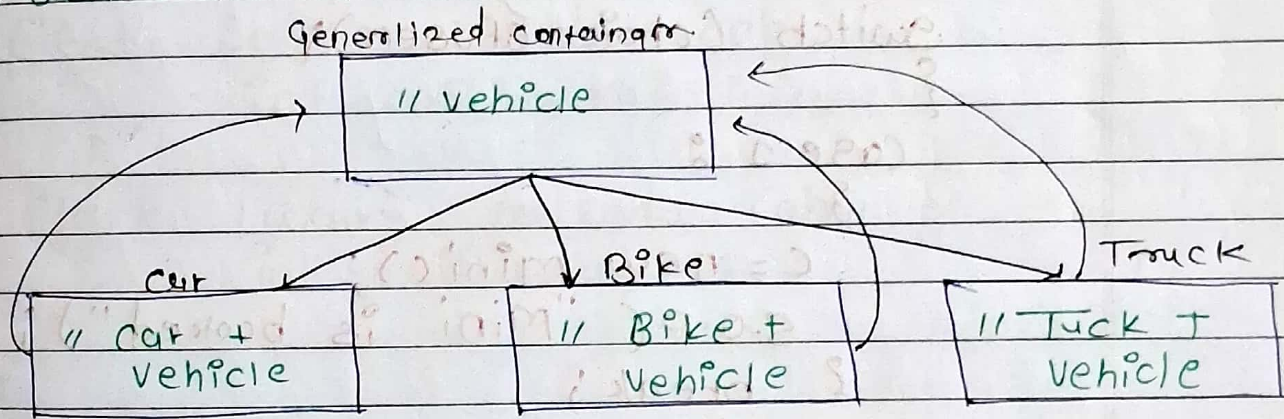


Purpose of upcasting :-

① Generalization
② Run-time Polymorphism

* Generalization :-

① Generalized container :-



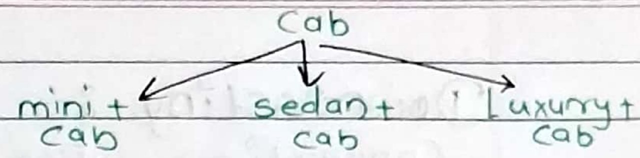
Vehicle v1 = new Car(); | vehicle v2 = new Bike(); | vehicle v3 = new Truck();

+ if we want to do upcasting we need inheritance
+ if we want to perform downcasting we need upcasting.

Inheritance ← upcasting ← Downcasting

Understanding of Generalization

```
Public class Main(){
  public static void main(String[] args)
  {
    S.O.P("Welcome to ola");
    S.O.P(" Enter 1 to select Mini");
    S.O.P("         2          sedan");
    S.O.P("         3          Luxury");
    Scanner sc = new Scanner();
    int choiceofuser = sc.nextInt();
    Cab c = null;
    switch (choice ofuser){
    {
      case 1 :
        {
          c= new mini();
          S.O.P( "Mini is booked");
        } break;
      case 2 :
        {
          c = new sedan();
          S.O.P("Sedan is Booked");
        }
        break;
      case 3:
        {
        C= new Luxury();
        S.O.P ("Luxury is booked");
        }
      default : "Invalid Number";
        } S.O.P ("Invalid Number");
      }
  } }
```

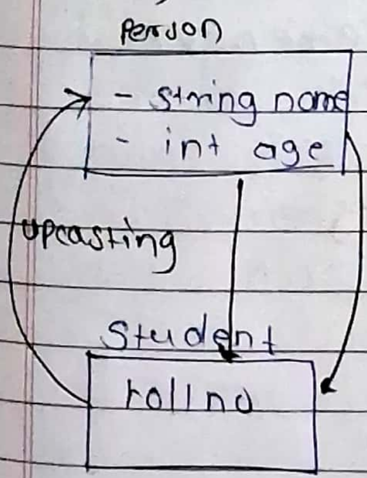



```
class Cab {
    int fare;
    String nameofDriver;
}

class mini extends Cab {
    int noofseats = 4;
}

class Sedan extends cab {
    int noofseats = 5;
}
class Luxury extends Cab {
    int noofseats = 6;
}
}
```

Note:-
After upcasting with Parent class variable we cannot access child specific member so, There is need a do downcasting




```
Person P = new Student();
S.o.p (P.name);   // null;
S.o.p (P.age);    // 0;
S.o.p (P.rollno); // CTE;
```

we cannot access rollno in with upcasting we need to do downcasting

downcasting is done by the Programmer by using typecast operator.

classmate
Date
Page

Downcasting :-

- Converting Parent type reference back to child type is known as downcasting
- Downcasting can be only done by programmer using type-cast operator

Purpose of downcasting

1. To access child class member we perform downcasting

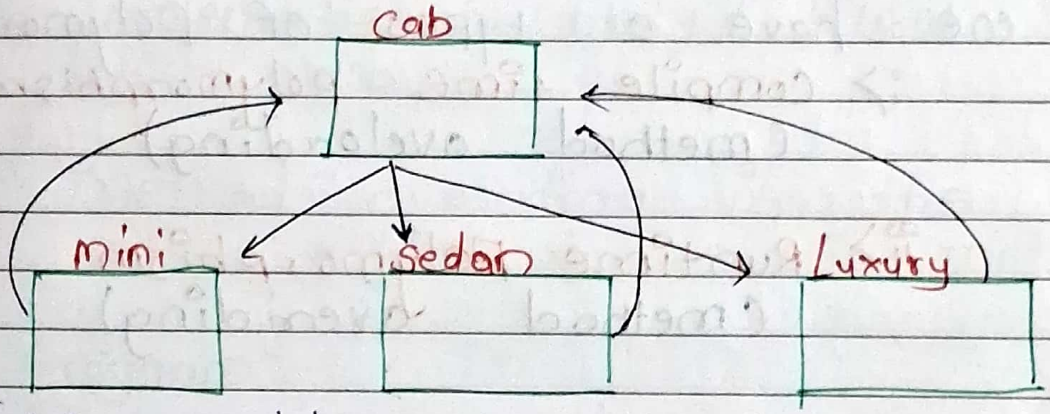
Program :-

```
Public class studenDriver
{
    // upcasting
    Person P = new Student();
    S.o.p(P.name);
    S.o.p(P.age);
    // cannot access child member
    // using parent reference

    // Downcasting
    Student s1 = (Student)P;
    S.o.p(s1.rollno);
}
}
class Person {
```

String name;
int age;
}
class Student extends Person
{
int rollno;
}

* Class Cast Exception :-



Cab c = new mini();
Sedan s = Sedan c; // ClassCastException.

When we perform upcasting on one child & doing downcasting on another child that timing we get ClassCastException

When we try to downcast the reference to a child type but the object does not have the instance or reference of that subclass. we will get classCastException.

classmate
Date
Page

* Polymorphism :-

① Poly means many where morphism means forms.

② When one object exhibit multiple behaviour in different scenarios is the concept of polymorphism

③ we have 2 types of Polymorphism
   i> compile time polymorphism (method overloading)
   ii> Runtime Polymorphism (method overriding)

① compile time ~~mot~~ polymorphism can be achieve by method overloading loading

② Method with the same name but different formal argument in the same class is known as method overloading

Runtime polymorphism :-

① Runtime polymorphism can be achieve by ~~or~~ method overriding

② Methods cleated / declared with same name , same formal argument but with different method implementation / body / block in the child class .

③ Method overiding is the concept only for Non-static methods.

Program :-

```
Package polymorphism;
Public class Driver {
    main() {
        Child c = new Child ()
        c.colorOfHouse (); → Lavender
        Parent P = new Parent();
        P.ColorOfHouse (); → Blue
        Parent P1 = new Child ();
        P1.ColorOfHouse (); → Lavender
    }
}
class Parent {
    Public void colorOfHouse () {
        S.O.P ("Blue");
    }
}
```

```
class Child extends Parent {
    Public void ColorOfHouse()
    {
        S.O.P ("Lavender");
    }
}
```

NOTE :-
Execution of the method implementation depends on the object created & does not depends on the type of reference used to ~~class~~ call.

* java.lang.Object :-

Object class :-

1. Class Object is the root of the Class hierarchy
2. In java, every class haves object as a super class or Parent class.
3. It is defined in Java.lang Package
4. It consist of 11 Non-static method which is inherited to all the subclasses.

Methods in Object Class

Protected object Clone()
public boolean equal (object o) ←
Protected void finalize()
public class getclass()
public int hashCode() ←
public void notify
public void notifyAll()
public String toString() ←
public void wait()
public void wait(long timeout)
public void wait(long timeout, int nanc))

* Public String toString() :-

It is used to return string representation of address or reference

String Format :
Classname@Hexadecimal Number.

The Purpose of overriding to string method to return the states of an object or Properties of an object not the address

Program:
Before overiding toString()

```
Public class Student
{
    String name;
    int age;
    int rollno;

  Public Student(String name, int age,
                 int rollno)
  {
      this.name = name;
      this.age = age;
      this.rollno = rollno;
  }
  main(String [] args)
  {
      Student s1 = new Student("Riya", 21, 41);
      Student s2 = new Student("Tanvi", 22, 32);
      S.o.P(s1); // Student@0x1
      S.o.P(s2); // Student@0x2
```

After overridding tostring()

```
Public class student
{
      String name;
      int age;
      int rollno;
```

controls to create variable of object

```
public student (String name, int age,
                int roll no)
{   this.name=name;
    this.age = age;
    this.rollno=rollno;
}
Public String toString()
{
    return name+" "+age+" "+rollno;
}
main (String [] args)
{
Student S1 = new Student ("Riya", 21, 41);
Student S2 = new Student("Tanvi", 22, 31);
S.o.p (S1);   // Riya 21 41
S.o.p(S2);    // Tanvi 22 31
```

* Public boolean equals (object o):- Comparision of two object using equals to operator

```
public class Marker
{
    String brand;
    String color;
    double Price;

    Public Marker (String brand, String color,
                   double price)
    {
        super();
        this.brand = brand;
```

this. Color = Color;
this. Price = Price;
}
main (String [] args)
{
Marker m1 = new Marker("Camlin", "Black", 50);
maker m2 = new Marker("Camlin", "Black", 50);
S.o.p(m1 == m2) // == operator compare the address
}

① equals method is used to compare the address of two objects
② The purpose of overiding equals method is to compare the states or Properties not the address.

Example :-

Comparison of 2 operatd object using equals method

```
Public class Marker
{
    String brand;
    String Color;
    double price;
  public Marker (String brand,
          String Color, double price)
    {
```

when we pass argument in the method declare that argument is known as formal argument

Date
Page

```
super();
this.brand = brand;
this.color = color;
this.Price = Price;
}
Public boolean equals(Object o)
{
  Marker m = (Marker) o;
  if (this.brand == m.brand, this.color == m.color, this.Price == m.price)
  {
      return true;
  }
  return false
}
public main()
{
  Marker m1 = new Marker("Camlin", "Black", 50);
  Marker m2 = new Marker("camlin", "Black", 50);
  S.o.p (m1.equals(m2)); it will compare the states/Properties.
}
```

Actual argument

Q) create class student with Properties name, age, rollno store two stundent objects and display their details. create two more object with the same Properties & compare them.

* Public int Hashcode();

1. this method is used to generate a random integer value for an object
2. HashCode() method generates the hashcode value / random integer value based on the address by using hash()
3. hash() is present in Objects class, which is defined in java.util Package

Purpose of overiding Hashcode()
The purpose of overiding Hashcode method is to generate random integer value based on the properties not the address.

Program:-

```
import java.util.Objects;
public class fruit {
    String name;
    String color;
    public fruit (String name, String color)
    {
        this.name = name;
        this.Color = color;
    }
    public String toString () {
        return name+" "+color;
    }
```

hashCode() → random integer value
hash() → address

```
public boolean equals(Object o)
{
    Fruit f = (Fruit) o;
    return this.name == f.name &&
           this.color == f.color;
}

public int hashCode()
{
    int res = Objects.hash(name, color);
}

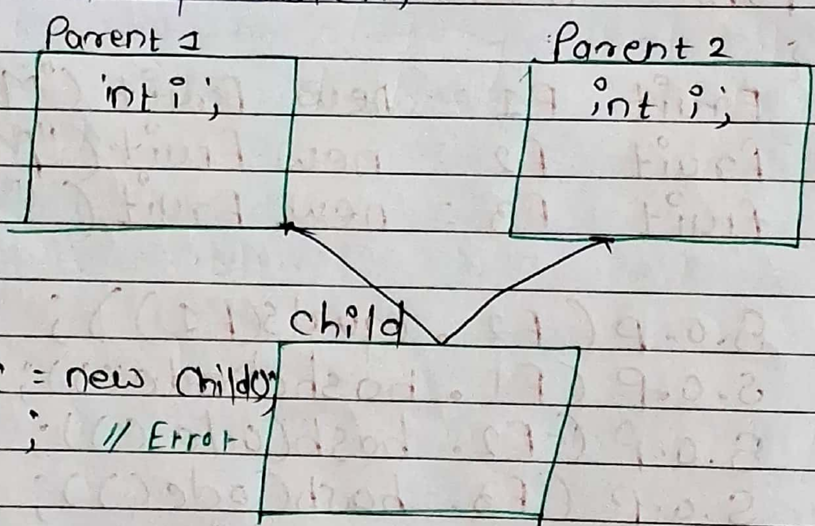
public main()
{
    Fruit F1 = new Fruit("Mango", "Y");
    fruit F2 = new Fruit("Mango", "Y");
    fruit F3 = new Fruit("Mango", "Y");

    S.O.P(F2.equals(F1));          // equals()
    S.O.P(F1.hashCode());          }
    S.O.P(F2.hashCode());          } // hashCode()
    S.O.P(F3.hashCode());          }
    S.O.P(F3.==F1);                // toString()
}
}
```

* Diamond Problem :-

① Multiple inheritance cannot be achieve by using class if we try do so we will get diagmond problem

② The ambiguity or confusion in multiple inheritance which arises when a super class member is try to call from child class this confusion is known as diamond Problem

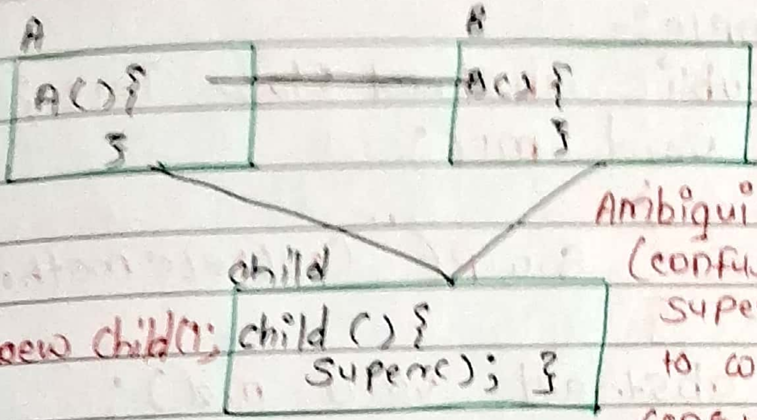


child c = new child();
(c.i); // Error

Defination :-

If we try to achieve multiple inheritance using class, when we create a object of child the child class constructors super() statement will get confused weather which parent class contructor it should call.

Diamond Problem.



child C = new child();

Ambiguity (confusion for super() statement to which Parent constructor we have to call).

* Abstraction :-
① Hiding the implementation & displaying just necessary details of an object is known abstraction.
② In java we can achieve abstraction by using abstract()me
③ In jave we have two abstract component.
 i) abstract class
 ii) Interface.

i) abstract Class :-
Any class created with abstract modifier is known as abstract class.
Example :-
public abstract class Test {
}
(public ↑ Access modifier; abstract ↑ modifier)

④ Inside abstract class we can Create abstract as well as Concrete method.

Example:-

```
Public abstract class Test {
    void m1()
    {
        S.o.p("Concrete method");
    }
    abstract void m2();
}
```

⑤ If any class consist atleast one abstract method it is mandatory to make a class as abstract.

⑥ It is not mandatory that abstract class should have abstract methods.

ex:-

```
Public abstract class Test
{
    P void m1()
    {   S.o.p("concrete Method");
    }
}
```

⑦ Abstract class cannot be instantiated i.e we cannot created object for abstract class

Ex:-

```
Public abstract class Test{
    abstract void m1() {
}
Class Demo{
```

```
Public static void main(String [] args)
{
    Test t = new Test();    // CTE
                 we cannot create
                 instantiate
}
```

Abstract Method :-

Any method declared / created with abstract keyword but without implemetation or method body is known as abstract method.

Ex :-

abstract void m1();

- child as abstract — instantiate not possible
- override the method in child — instantiation possible.

*** Inheritance using abstract Class**

When we inherit abstract we have 2 option

i> we should make child class abstract.

ii> We have to overide all the abstract method of abstract class in child class

classmate
Date
Page

* INTERFACE :-

① In java, it is an abstract component (incomplete component)
② It is a component like a class
③ It is used to achieve 100% abstraction before Java 8

'How to create an interface
Syntax :- interface identifier {

}

④ interface is a keyword
⑤ interface can have only Public static final variables and Public abstract method

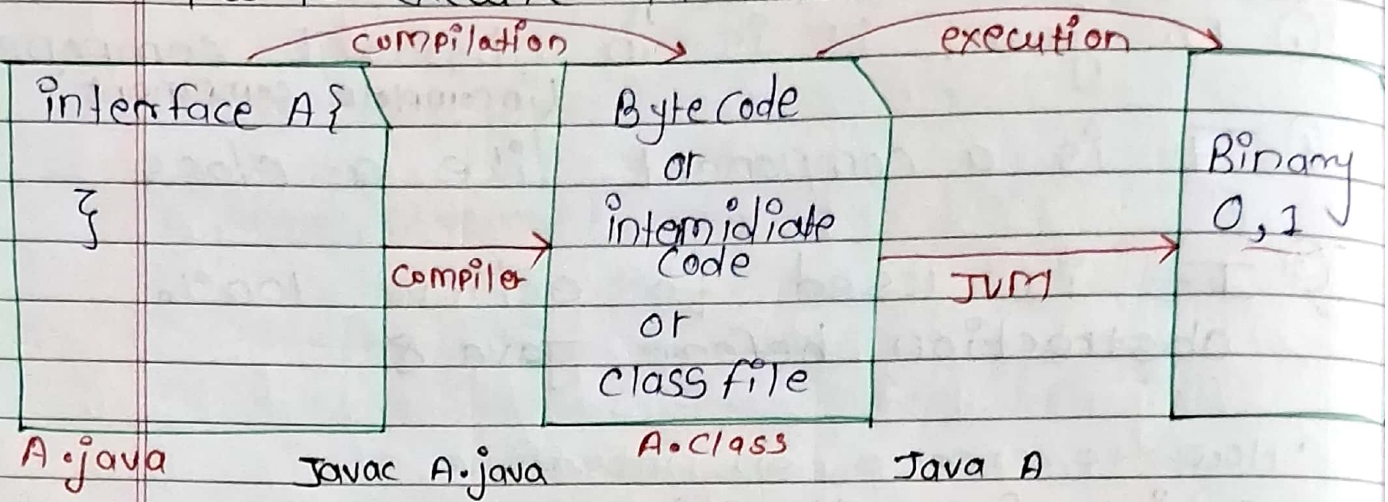
NOTE :-
if programmer fails to add the modifier automatically compiler will add public static final with variable & public abstract with method.

Program :- // Before Compilation

```
public interface Demo {
    int a=10;
    void m1();
// After Compilation
public interface Demo {
    public static final int a = 10;
    public abstract void m1();
}
```

this case till Java 7

How to execute an interface?



* NOTE :-

① Till Java 7 interface is used to achieve 100% abstraction as well as Multiple Inheritance

② from Java 8, we can create static & default method also inside an interface.

from Java 8 → Hence we cannot achieve 100% of abstraction

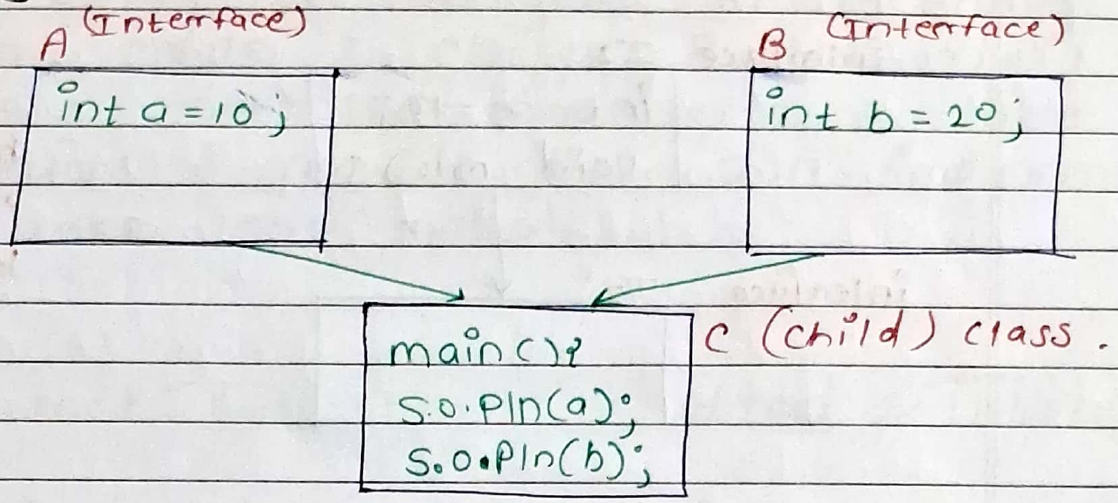
```
interface identifier{
    static + default methods
    public static final variable,
    public abstract method;
}
```

NOTE :- static method cannot be abstract.

Class to Class
Interface to Interface } extends

Class to Interface → implements.

* Inheritance Between an interface & class



class child_classname implements A, B.

* NOTE :-

If we perform inheritance between an ~~inh~~ interface & class we should use implements keyword

For example :-

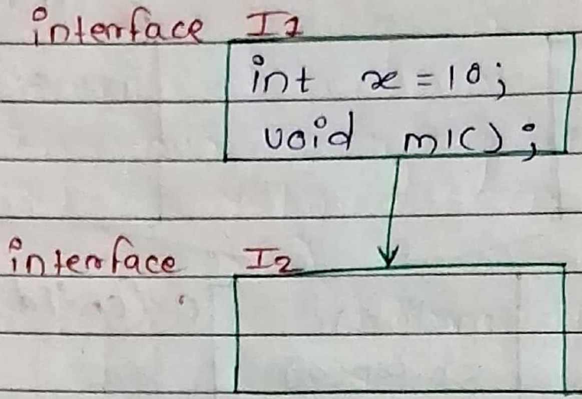
class child-class-name implements interface_name

Program :-

```
Public class child implements A, B {
    public static void main (string [] args) {
        S.O.P (a);
        S.O.P (b)}
    }
}
interface A {    int    a= 20;    }

interface B {    int    b= 80;    }
```

* Inheritance bet^n interface & interface



here we use extends keyword.

NOTE :-
If we perform inheritance bet^n interface & interface we should use extends keyword.

example :-

```
public interface I1{
        int x=10;
        void main();
  }
public interface I2 extends I1 {
        int y=20;
    public static void main (String [] args)
     {
        S.o.p(x);
        S.o.p (y);
     }
}
```

classmate
Date
Page

Q> WJP to create an instance (object) interface sortable with a method sort that sort the array of intergers in ascending order. create 2 classes Bubblesort & selection sort that implements sortable interface & provide their own implementat for the sort method.

N
* Difference Between Abstract & Interface

abstract	interface
① It is class which consist of abstraction and concrete methods.	interface is a 100% abstract component which can consist only public static final var + public abstract method
② we cannot create object but we can create constructor for abstract clas	we cannot create an ~~ab~~ object as well as constructor.
	To support multiple inheritance.
③ it create with abstract keyword	it create with interface keyword.

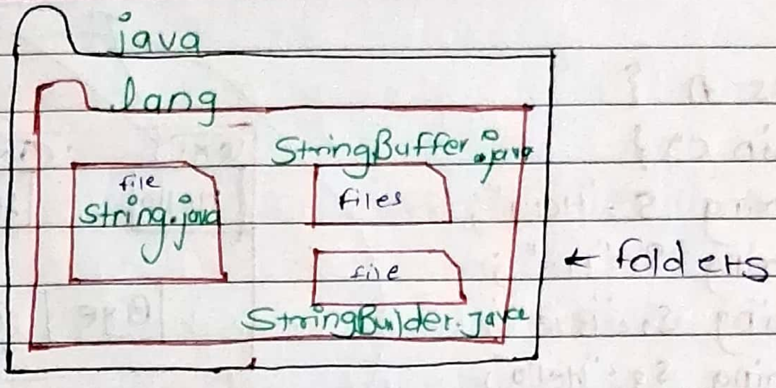
* String :-
① It is a collection of character
② every string in java must be enclosed within " " quotes
③ In java string is class
④ It is defined in Java.lang (No need to import)
⑤ why? To store group of collection

How to store?
i> java.~~util~~ lang.String
ii> java.~~util~~ lang.StringBuffer
iii> java.~~util~~ lang.StringBuilder

⑥ To use String in java object should be created by using following String, StringBuffer, StringBuilder.

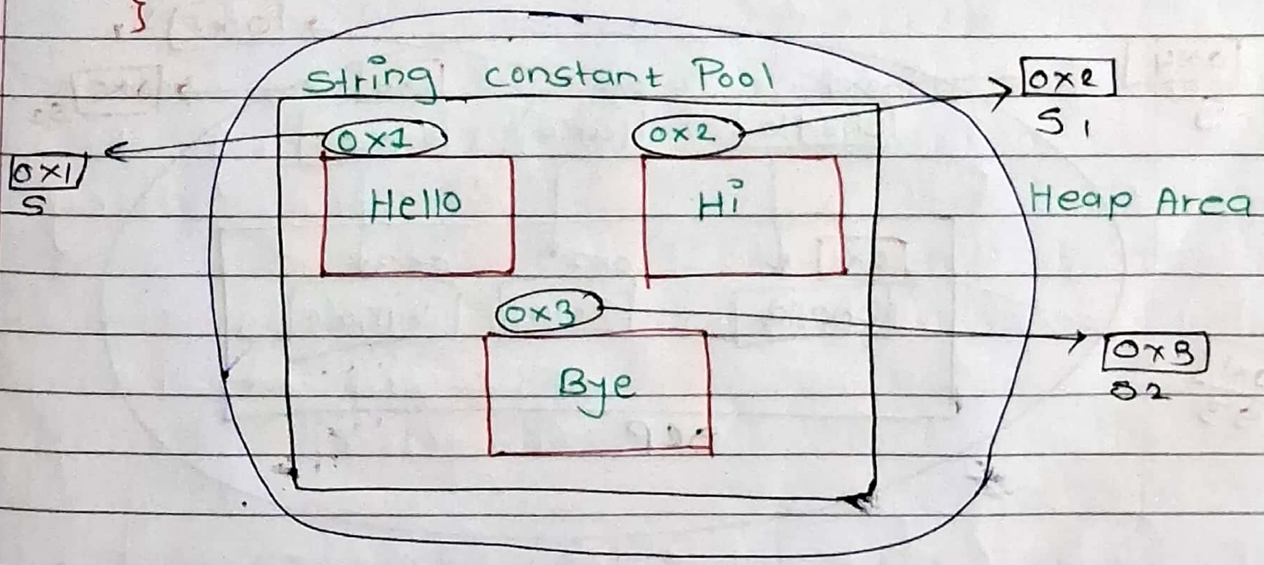
NOTE :-
① When a programmer doesnot create an object to used the String by default for every ~~Strin~~ unique string literal one object is created in String Constant Pool.

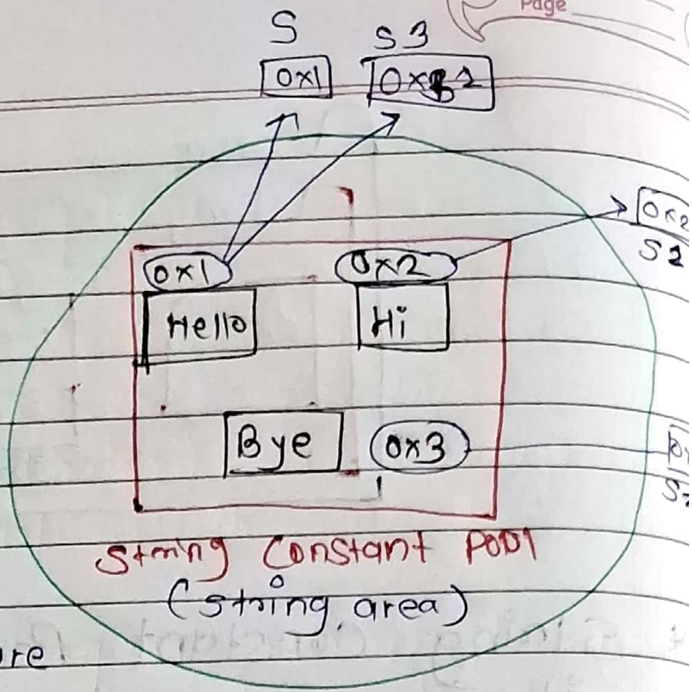
② we should not import the files which are present Java.lang folder.



* String constant Pool :-
it is an area located inside the heap area used to store string literal object.

```
class A {
main() {
String s = "Hello";
String s1 = "Hi";
String s2 = "Bye";
}
}
```



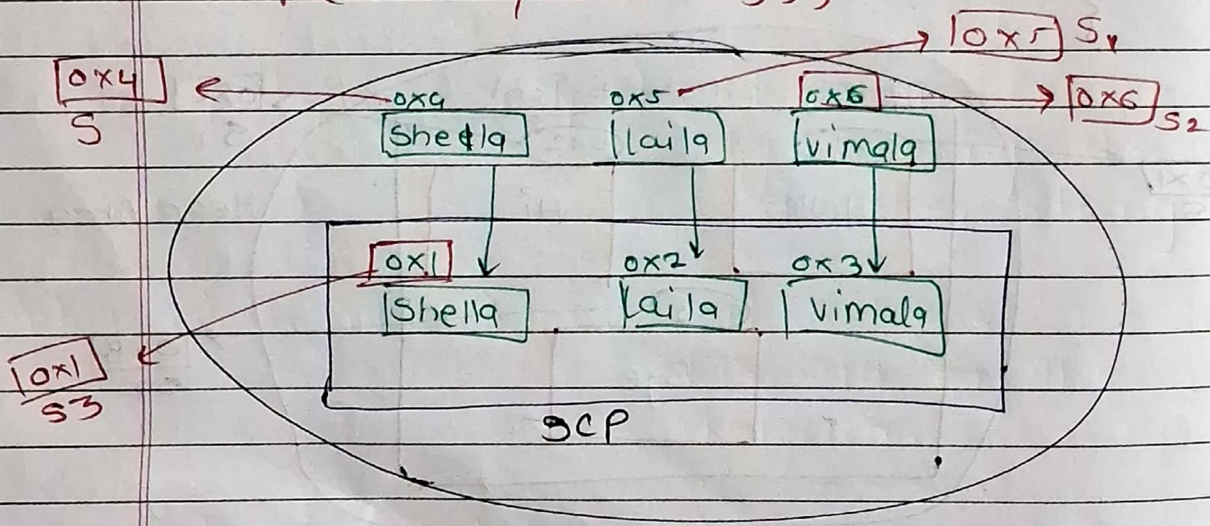



```
Class A {
 main () {
  String S = "Hello";
  String S1 = "Hi";
  String S2 = "Bye";
  String S3 = "Hello";
  S.o.P(S==S3);
}
```

== operator compare address

Program - 3

```
String S = new("Shella");
String S1 = new("Laila");
String S2 = new("Vimala");
String S3 = "Shella");
S.o.P( S == S3 )
S.o.P( S. equals(S3));
```



S==S3

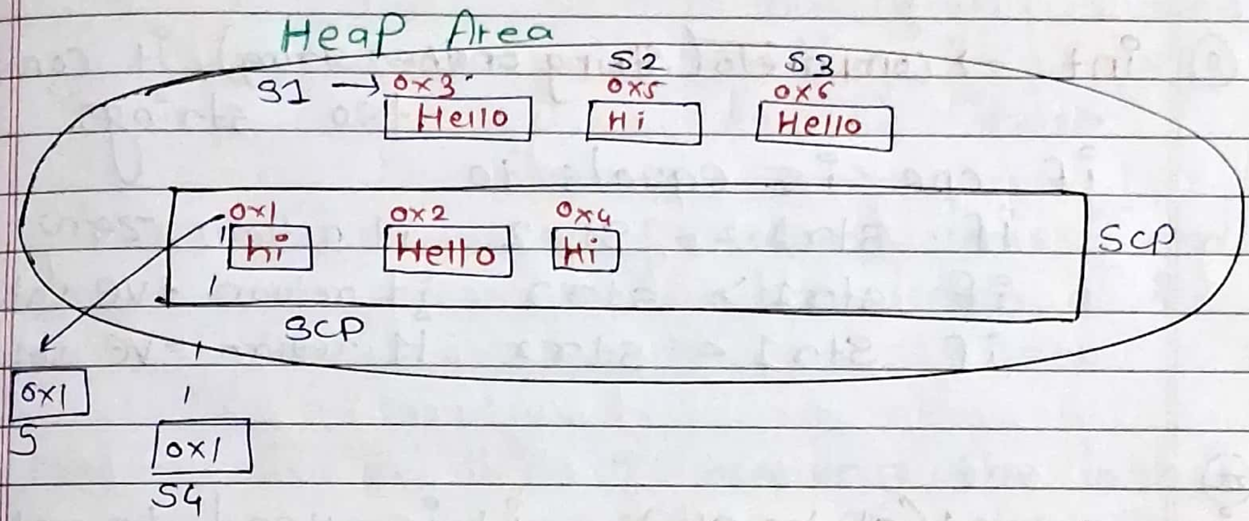
0x4 = = 0x1

S. equals(S3)

Date
Page

Program 4 :-

```
String  s  = "hi";
String  s1 = new String("Hello");
        s2 =     String(" Hi ");
        s3 =           ("Hello");
String  s4 = s;
```




```
S.O.P (s1.equals(s3));  // true
      (s == s1);         // false
      (s == s4);         // true
      (s == s2);         // false
      (s.equals(s1));    // false
      (s.equals(s4));    // True
      (s.equals(s3));    // false
      (s == s3);         // false
```

Method in String class :

returntype | method signature | Description

① char → charAt(int index) It returns charac present at the specified index

② int → compareTo(String anotherString) it compare two strings.

if one is equals to
if str1 == str2 it return zero.
if str1 > str2 it return +ve value
if str1 < str2 it return -ve value.

③ String concat(String str) it is used to return the combination of two strings

boolean equals(Object anObject) it returns boolean value by comparing the states of the string

int indexOf(int ch) it returns the index of ~~Present at~~ the given Character.

int indexOf(int ch, int fromIndex) it returns the index of the character by search from the specific index in the string

int	indexof(String str)	it returns the index of the string present in the given string
int	indexOf(string str, int fromIndex)	It returns the index of the string present in the given string by searching it from a specific index
boolean	isEmpty()	It returns true if and only if the length of the string is zero
int	lastIndexOf(int ch)	It returns the last index of the character present in the string
int	lastIndexOf(int ch, int fromIndex)	It returns the last index or last occurance of the character present in the string By searching it from a specified index.
int	lastIndex(String str)	It returns the last occurance of the string present in the given string

Int	lastIndex(string str, int fromIndex)	It returns the last inde of the String present in the given string b searching it from the specified index.
int	length()	It returns the number of character present in the string.
String	replace(~~old~~ char old char, char new char)	It returns a strings by replacing all the old charater from the new character.
String[]	split(String regex)	It returns string array by spliting the string based on a string.
String	substring(int beginIndex)	It returns thestring from the specific index till the end of string
String	substring(int beginIndex, int endIndex)	It returns the part of the string from the given string from starting to the end index.
Char []	toCharArray()	It returns the character array ~~to~~ by converting the string into character array

length / length()

→ variable
→ is used to check the length of ARRAY

|→ Method
|→ it is used to check the length of STRING

classmate
Date
Page

String toLowerCase() — It ~~returns~~ converts the lowercase character of the string

String toUpperCase() — It convert all the character to uppercase.

String ~~for~~ trim() — It is used remove all the leading spaces in the string i.e ~~forward~~ & starting & ending space.

NOTE :-

① STRING is Immutable in nature
immutable means unchangable.
Example :-

```
Public class Example {
    public static void main(String [] args){
        String s = "SHEELA";
        s.toLowerCase (); this is only used to convert the string but cannot store in
        S.O.Pln ( s ); // s object is immutable
```

② StringBuffer & StringBuilder are mutable in nature.
Example :-

```
Public class Example {
Public static void main (String [] args) {
    StringBuffer sb = new StringBuffer(" sheela");
    StringBuilder Sb1 = new StringBuilder("Laila");
    Sb. reverse();
    S.O.P(sb);
    Sb1. reverse();
    S.O.P(sb1);
} }
```

* final keyword :-

1.
2. It is non-access modifier
3. final can be used with variable, method and class.
4.

* final variable :

1. if any variable created or declared with final keyword is known final variable.

2. final variable cannot be reinitialize if we try do so we will get CTE (compile Time Error)

Program

```
Public class Example {
    Public static void main (String [] args)
    {
        final int a = 10;
            a+= 100;        // CTE
    }
}
```

* final method :-

① Any method created or declared with final keyword is called final method

② final method cannot be overridden in the child class.

③ For example :-

```
class A {
    public final void display() {
        S.o.p("final method of A");
    }
}
Class B extend A {            // CTE
        public void display() {
            //
        }
}
```

* final class :-

① If any class is final it cannot be written super class or Parent class

② String & Scanner are final in nature (Java)

```
final class A {

}
class B extends A {
                        } CTE
}
```

Collection

* Collesction framework :-

① Framework :-
The group of multiple classes & interfaces is known as framework

Collection :-
① It is group of objects
② In java, collection is a interface

Why collection?
① To store multiple objects we need collection
② To overcome the disadvantages of array we need collection
→ only homogeneous data
→ size of array is fixed on runtime
③ we can store heterogeneous data inside collection
④ The size of the collection is dynamic

* NOTE :-
Every object in collection is upcasted to java.lang.object type

collection is created inside the Java.util Package

** Collection Hierarchy **

Collection(I)

- List(I)
  - Arraylist
  - LinkedList
  - Stack
- Set(I)
  - HastSet
  - LinkedHashSet
  - TreeSet
- Queue(I)
  - priority queue
  - ArrayDeque

* java.util.collection :-

① It is an interface in Java
② It is defined in java.util Package
③ It consist of ~~subome~~ some abstract methods which are used to Perform CRUD operation
④ Collection has a Parent java.util.Interable

* Methods in collection

boolean	Adding	add (object o);
boolean		addAll (~~object o~~ Collection c);

classmate
Date
Page

return type	Method name	
boolean	add (object o);	Adding
boolean	add (collection c);	
Iterator	iterotor( );	Accessing
boolean	contains(object o);	Searching
boolean	containsAll(collection c);	
void	Clear ( );	Removing
boolean	remove (object o);	
boolean	removeAll (collection c);	
boolean	retainAll (collection c);	

⇒ micellaneous

size( );
equals (object o);
hashcode( );
isempty( );

classmate
Date
Page

* List :-
1. It is an Interface
2. It is defined in java.util Package.

characteristics of list :-
1. NULL is accepted
2. Insertion order is maintained
3. Duplicate object are allowed
4. Heterogeneous object are allowed
5. Indexing is possible

Method in List :-

1. Adding
   - add(object o); } inherited from collection
   - addAll(collection c); } inherited from collection
   - addAll(int index, object o);
   - addAll(int index, collection c);

2. Accessing
   - iterator(); → inherited
   - get(int index);
   - listIterator();
   - listIterator(int Index);

3. Searching
   - contain(object o); } inherited
   - containAll(Collection c); } inherited
   - index of(object o);

4. Removing
   - clear(); } inherited
   - remove(object o); } inherited
   - removeAll(Collection c); } inherited
   - remove(int index);
   - retainAll(collection c); → inherited
   - ~~isEmpl~~

classmate
Date
Page

⑤ micellaneous size();
equals (Object o);
hashCode ();
isEmpty();

* ARRAYLIST :-
java.util.ArrayList
① It is a class
② it is a subclass of List Interface
③ it stores the objects in the dynamic array

Note:-
① when we create an object for arraylist one array is created inside the memory of size 10

② Based on the number of object we are adding & load factor the size of the array will increased by the formula:-

New capacity = (Current capacity x {(3/2) + 1})

characteristics of Arraylist:-
① Null is accepted (Any number of null)
② insertion order is maintained
③ Duplicated objects are allowed
④ Heterogenous objects are allowed
⑤ Indexing is possible

New capacity = (Current capacity × $\frac{3}{2}$) + 1

classmate
Date
Page

addAll() → it is used to transfer (copying) one collection into another collection.

contains() :- check whether the object is present inside the collection or not.

Clear() :- method is used to remove all the object in the boolean.

remove() :- method is used to remove that perticular object & return the boolean value.

removeAll() :- it is used to remove only (common) some objects in the collection

retainAll :- it is used to store only (common) same object in the collection.

NOTE :-
① Arraylist Follows Dynamic Array or growable size array.

load factor = 0.75 × 10
= 7.5

classmate
Date
Page

① Adding the object in the collection.

Example:

```
import java.util.ArrayList;
main()
{
    ArrayList BAG1 = new ARrrayList();
    BAG1.add("Mango");
    BAG1.add("kiwi");
    BAG1.add("Apple");

    Arraylist BAG2 = new Arraylist();

    BAG2.add("Pineapple");
    BAG2.add("Grapes");
    BAG2.add("chikoo");

    BAG1.addAll(BAG2);
// Adding all the object from bag2 to bag1
    S.O.P(BAG1);
// [mango, kiwi, Apple, pineapple, grapes, chiakoo]
    S.O.P(BAG2);
// [pineapple, Grapes, Chickoo]
```

classmate
Date
Page

② searching the objects in the collection

```
import java.util.Arraylist;
class ALEX {
  main ()
   {
   ArrayList Bag1 = new Arraylist();
     Bag1.add(" Mango");
     Bag1.add(" kiwi ");
     Bag1.add(" APPle ");

  Arraylist Bag2 = new ArrayList();
    Bag2.add("Pineapple");
    Bag2.add(" chickoo ");
    Bag2.add(" Grape");
    Bag2.add(" mango ");
    Bag2 add(" kiwi");
    Bag2 add(" Apple");

   S.o.P( Bag1. Contains("chickoo"));
 // False
   S.o.P( Bag2.containsAll ( Bag1));
   // True
   S.o.P( Bag2.indexALL("mango"));
   // 3
```

③ Removing the objects from the Collection

```
import java.util.Arraylist;

Class ALEx1
{
  main ()
   {
   Arraylist Bag 1 = new Arraylist();

    Bag1. add (" mango ");
    Bag1 .add (" kiwi" );
    Bag2. add (" Apple ");

  Arraylist Bag2 = new Arraylist ();

   Bag2 . add (" Pineapple ");
   Bag2 . add(" Grapes ");
   Bag2. add (" chickoo ");

   Bag1 . clear()
// It will remove all the abject from the collect^n
   S.O.P (Bag1 );
   // [ ]   ← empty list
   S.O.P ( Bag2 . remove ("Banana"));
      // False
   S.O.P (Bag2 );
   //[Pineapple, Grapes, chickoo]
```

(4) RemoveAll() & retainAll() methods

```
main()
{
  Arraylist bag1 = new Arraylist();
  bag1.add("Mango");
  bag1.add("kiwi");
  bag1.add("Apple");
  bag1.add("Pineapple");

Arraylist bag2 = new Arraylist();

 bag2.add("Pineapple");
 bag2.add("Grape");
 bag2.add("Chickoo");

 bag1.removeAll(bag2);
// It removes the common objects
 S.o.p(bag1);
 [Mango, kiwi, Apple]
 bag1.retainAll(bag2);
// It stores the common objects
 S.o.p(bag2);
 [Pineapple]
```

next() → return the object
→ move the cursor to the next object.

* Java.util.Iterator :-

1. It is an Iterface
2. It is defined in java.util Package
3. It consist of three methods
   i) next()
   ii) hasNext()
   iii) remove()

i) next() :-
next() will return the object pointed by the cursor & will move the cursor to the next object

ii) hasNext()
It is used to check to whether the cursor is pointing to any object or not.

iii) remove() :-
it is used to remove the object while accessing.

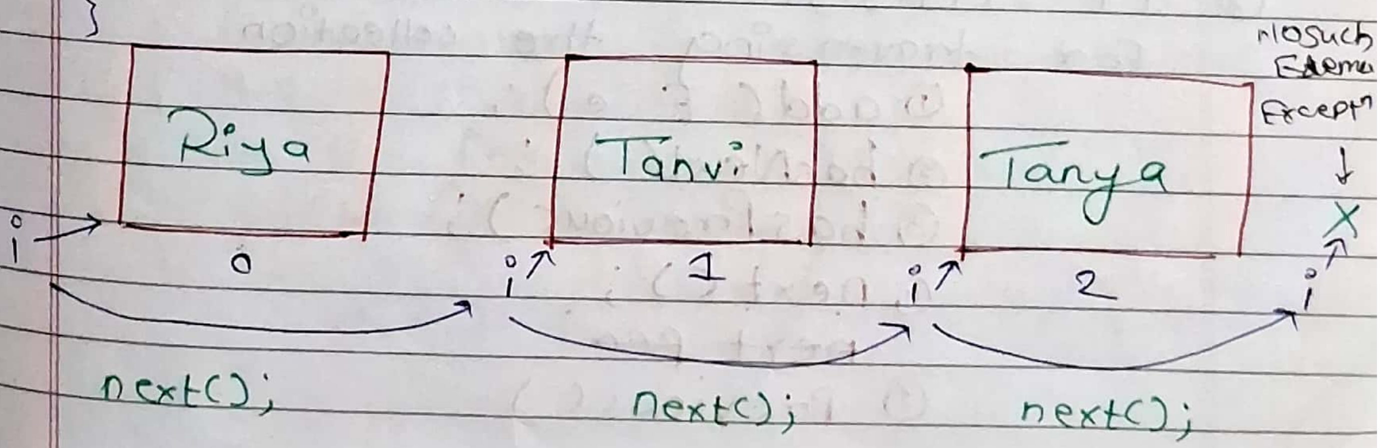
iterator (I)
↓
EventReaderDelgate (class)

* Accessing the object of collection using iterator

```
import java.util.Arraylist;
import java.util.Iterator;

class AccessingOperation
{
   main()
   {
      Arraylist al = new Arraylist();
      al.add("Riya");
      al.add("Tanvi");
      al.add("RTanya");

      Iterator i = al.Iterator();
      s.o.p(i.next());
      // Riya
      s.o.p(i.next());
      // Tanvi
      s.o.p(i.next());
      // Tanya
      s.o.p(i.next());
      // NoSuchElementException
   }
}
```



* Disadvantages of Iterator

1. we cannot traverse backward direction using iterator
2. Traversing more than once is not posible.
3. Traversing from specific index is not possible.

NOTE :-
To overcome the disadvantages of iterator ListIterator is used.

* java.util.ListIterator;

1. It is an Interface
2. It is defined in java.util Package.
3. It consist of some methods used for traversing the collection
   1. add(E e);
   2. hasNext( );
   3. hasPreviou( );
   4. next( );
      ~~next pea~~
   5. Previous( )

classmate
Date
Page

Q) Accessing objects using ListIterator. in Backward direction.

```
class Book {
   String title;
    double Price;
  public Book( String title ,double Price)
   {
        this.title = title;
         this.Price = Price;
   }
  Public String toString() {
       return title + " " +Price;
  }

import java.util.ArrayList;
import java.util.ListIterator;

Public class AccessListIterator {
    main ( String [] args) {
     ArrayList bag = new ArrayList ();
     bag.add (new Book ("Wings of fire", 350);
     bag.add (new Book("Godan", 300);
     bag.add (new Book("Harry Potter", 600);
     ba
     ListIterator li = new bag.ListIterator();
     S.O.P( li.next());
     S.O.P(" Forward direction : ");
     while (li. hasNext())
     {
           S.O.P (li.next()+"  ");
     }
```

```
System.out.Println();
s.o.p("Backward direction : ");

while (li.hasPrevious())
{
    s.o.p(li.Previous() + " ");
}
}
```

Program :

1. Create a class Product with Private instance variable Product_id, Product_name, Product_Price. Provide a getters & seters method to access the Properties

create a class shoes with Private instance variable shoe_size, Provide getter & setter for shoe_size. & inherit all the member of Product.

Create a class ~~customer~~ store & create a collection of 5 shoes of & store its details Print them.

classmate

Date
Page

For-each Loop :-

1. It is also known as Enhanced For loop
2. syntax - for each loop

```
for ( Object identifier : array/collection)
{
        // Instruction;
}
```

1. Example :-

```
import   java.util.ArrayList;
import   java.util.ListIterator;

Public class AccessingUsingforeach {
    public static void main (String [] args)
    {
        Arraylist al = new Arraylist();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        al.add(50);

        for(object o : al)
        {
            S.o.p(ob);
        }
    }
}
```

classmate
Date
Page

② ~~Exam~~
Working of for each loop

① For-each is only applicable for Array or collection

② The number of iteration in for each loop is equals to the number of object / elements present inside the collection/array

③ The type of variable declared isn for-each loop should be the same type of element/objects in the array or collection

④ For each iteration for each loop loop take one element from the ~~are~~ array or collection & store it inside variable declaration

Example :-

```
public class Exp1
{
    main ()
    {
        int [] arr = new int [5];
        for( int o : arr)
        {
            S.o.p ( o );
        }
```

output :-
0
0
0
0
0

LinkedList

* java.util.LinkedList :-

① It is a class

② It is subclass of List Interface

③ It is defined in java.util Package

④ linked list stores the object in the form of NODE

⑤ It follows Doubly linkedlist
A node which has two addresses

NODE :-
Combination of address and Data(obj

Previous Node address	data	Next Node Address

Node

* characteristics of LinkedList

① Ⓝ Null is accepted (Any No.of Null)

② Ⓘ Insertion order is maintained

③ Ⓓ Duplicated object are allowed

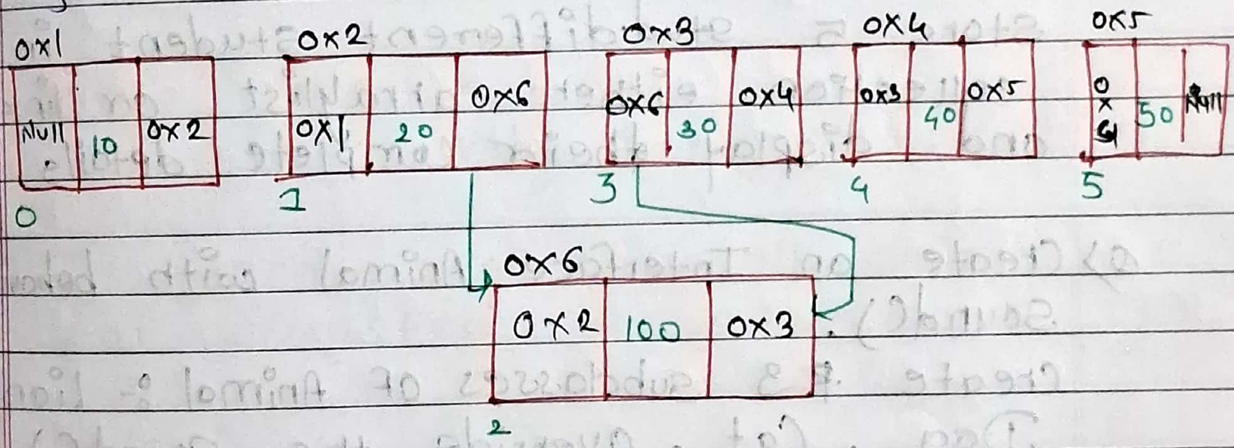
④ Ⓗ Heterogeneous object are allowed

⑤ Ⓘ Indexing is possible.

it store object in the from of node.

Example:

```
class LinkedListExample
{
   main()
   {
      LinkedList ll = new LinkedList();
      ll.add(10);
      ll.add(20);
      ll.add(30);
      ll.add(40);
      ll.add(50);
      ll.add(2,100);
   }
}
```



Output:
[10, 20, 100, 30, 40, 50] → Bcz toString method overidden.

* Types of collection

In Java we have 2 types of collection

i) Generic
ii) Non Generic collection

Non-Generic Collection :-

① Non-Generic collection stores heterogeneous objects

② All the objects will be updated to Java.util.object;

Generic collection :-

① It stores only homogeneous object

② In generic collection all objects are not upcasted to java.util.object type

Syntax :-

Classname <Non-Primitive Datatype> identifier = new constructor<>();

Example :

```
import java.util.Linkedlist;
class Book
{
    String title;
```

Date
Page

double Price;

Public Book (string title, double Price)
{
    Super();
    this.title = title;
    this.Price = Price;
}

class GenericExample
{
    main()
    {
        ~~Arraylist al =~~
LinkedArraylist < Book > al = new ~~Arraylist~~ Linkedlist <>();

        al.add(new Book("Godan", 350));
        al.add (new Book ("wing of fire", 400);
    }
}

Stack

java.util.Stack;

① It is a class

② defined in Java.util. Package

③ It Stores the objects in LIFO order
FILO

methods in Stack :-

① push(object o); → it is used to add the object in the stack.

② POP(); :- → It is used to remove the top most object from Stack.

③ peek(); → It is used to return tomost object from the stack

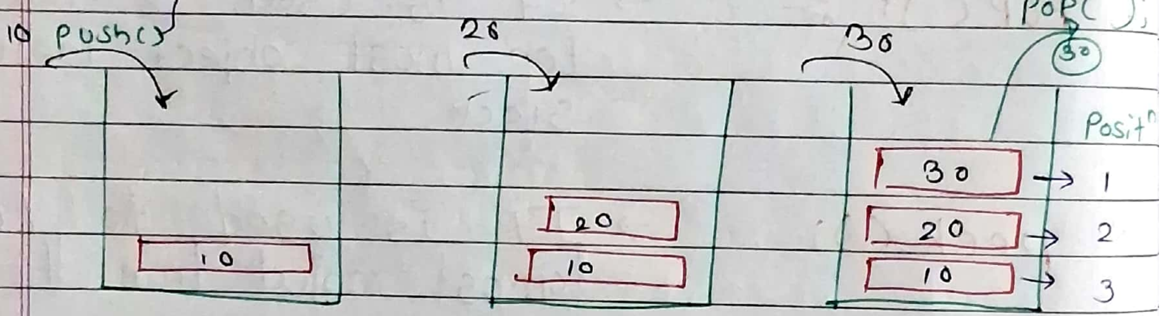
④ search(object o); → It is used to return the Position of object in stack (Strating from 1)

example :-

```
import java.util.Stack;

class StackExam
  {
     main()
    {
      Stack s = new stack();
      s.push(10);
      s.push(20);
      s.push(30);

      s.o.p(s);   // [10, 20, 30
   }
```




```
    s.o.p
    s.pop();
    s.o.p(s);  // [10, 20]

   s.o.p( s.Search (20));   // 2
```

classmate

Date
Page

(7)

```
main()
{
    Stack S = new Stack();
        S.Push(10);
        S.Push(20);
        S.Push(30);
        S.Push(40);
        S.O.P( S.POP()); //[10,20,30]
        S.O.P ( S.Search(20)); // 2
}
```

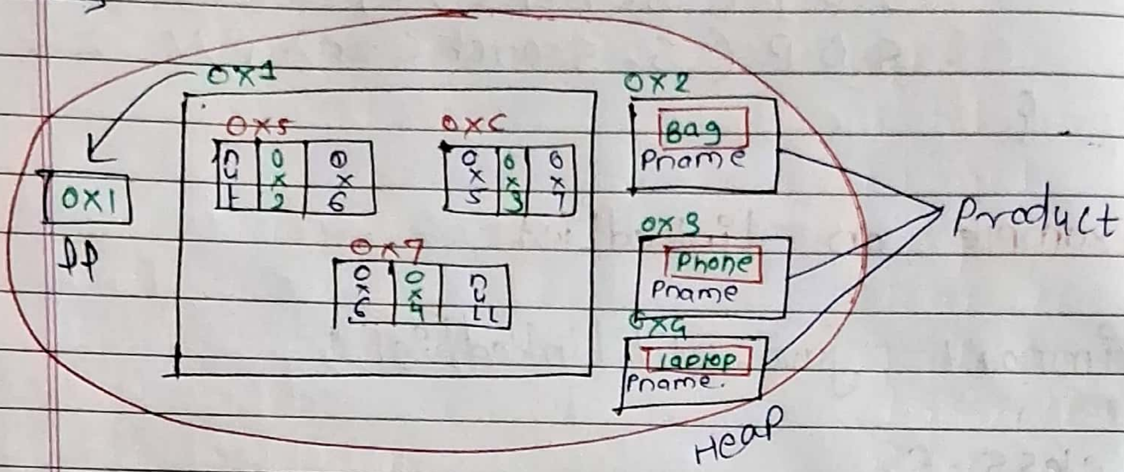
Example on LinkedList

```
import java.util.LinkedList;

class Ex
{
    main()
    {
        LinkedList ll = new LinkedList();

        ll.add(new Product("Bag"));
        ll.add(new Product("Phone"));
        ll.add(new Product("Laptop"));
        S.O.P(ll);
}
class Product
{
    String Pname;
        public Product (String Pname)
        {
```

```
Super();
 this. Pname = Pname;
}
public String tostring(String Pname)
{
  return Pname;
}
}
```



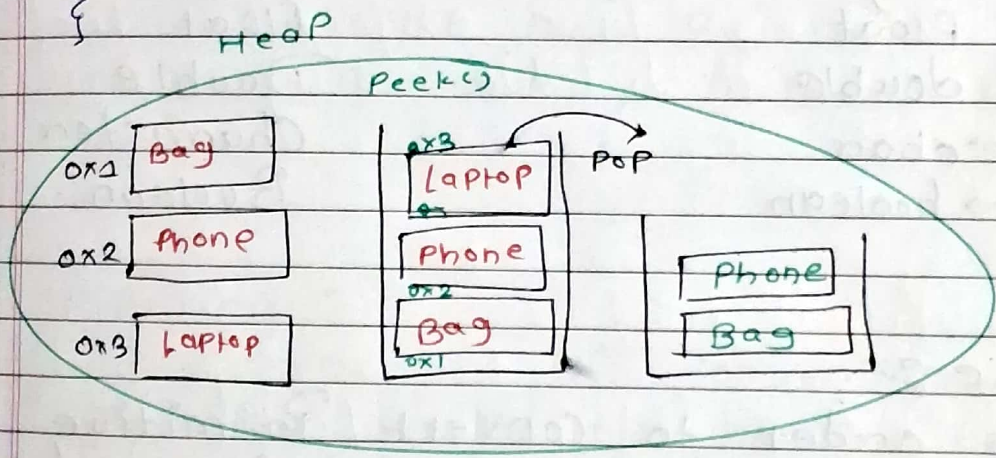
Example of Stack

```
import Java.util.Stack;
Public Class Ex
 {
  main()
  {
    Stack s = new Stack();
    s.Push(new Product("Bag"));
    s.Push(new Product("phone"));
    s.Push(new Product("laptop"));

    s.o.p(s.Pop()); // Laptop
    s.o.p(s); [Bag, Phone]
  }
```

```
class Product
{
    String Pname;

    $ Product ( String Pname ) :
    {
        Super();
        this.pname = Pname;
    }
    public String toString()
    {
        return Pname;
    }
}
```



* Wrapper Classes :-
This are the classes which is use confovert Primitive datatype to non-Primitive datatype & Non-Primitive datatype back to Primitive datatype.

Primitive Datatypes		~~Non-Primitive Datat~~ Class
integer	byte	Byte
	short	Short
	int	Integer
	long	Long
decimal Point	float	Float
	double	Double
character →	char	Character
either True or False →	boolean	Boolean

Note :-
① In order to convert Primitive datatypes to Non Primitive datatyp respective Classes are created for 8 primitive datatype all classe are known as Wrapper class

* Boxing :-
The Process of wrapping Primitiv to Non-primitive datatype is known as Boxing

~~The Process~~

The Process of ~~B~~ performing Boxing automatically is called as AutoBoxing

Example :-

```
Public class wrapper
{
   main ( )
   {
      int a = 10;
      // AutoBoxing
      Integer b = a;
      S.O.P (b);
   }
}
```

* UnBoxing :-

The process of converting Non-primitive datatype back to primitive datatype is called as Unboxing

The process of performing unboxing automatically is called as Autounboxing.

Exam :-

```
class ~~unb~~ wrapper
{  main () {
      Double obj = 3.14;
      // Auto Unboxing
      double d = obj;
      S.O.P(d);
   }
}
```

Scanned with OKEN Scanner

Set

* **Set :-**
java.util.Set;

1. It is an interface
2. It is defined in java.util.Package
3. The subclass of set are
   HashSet
   LinkedHashSet
   TreeSet
4. Set stores the object in the form of Hash Table using Hashcode value of an object.

* **java.util.Hashset :-**

1. It is a class defined in Java.util Package.

Characteristics of Hashset

1. only one Null is accepted
2. Insertion order isn't maintained
3. Duplicate objects are not allowed
4. Heterogeneous object are allowed

In hashcode method we generate Random integer value by using Hash() method which is a static method & called as a classname as a Reference.

classmate Date ____ Page ____

⑤ Indexing is not Possible.

Program

```
import java.util.HashSet;

Public class Example
{
    main ()
    {
        hs. add
        Hashset hs = new HashSet();
        hs.add("RCB");
        hs.add(" CSK");
        hs.add(100);
        hs.add("MI");
        hs.add(NULL);
        hs.add(NULL);
        s.o.p(hs);
    }
} // (RCB, MI, CSR, 100, NULL)
```

classmate
Date
Page

* java.util.LinkedHashSet

① it is a class defined in Java.util Package

② it is a subclass of # Set interf

Characteristics

① Null is accepted (only one)

② Insertion order is maintained

③ Duplicate object are not allowed

④ Heterogeneous objects are allowed

⑤ Indexing is not Possible

Program

```
import java.util.LinkedHashset;

  class Exam
    {
        main()
      {
         LinkedHashSet lhs = new LinkedHashSet
            lhs.add("RCB");
            lhs.add("CSK");
```

```
lhs.add(100);
lhs.add("MI");
lhs.add(NUll);
lhs.add(NUll);
S.o.p(lhs);
}
}        // [RCB, CSK, 100, MI, null]
```

* Java.util.TreeSet :-

① it is a class defined in java.util Package

② it is a subclass of set interface

Characteristics of TreeSet -

① Null isn't accepted (NO NULL)

② Insertion order is maintained (ascending order)

③ Duplicate object aren't allowed

④ Heterogeneous object aren't allowed

⑤ Indexing ~~no~~ is not possible.

Program :-

```
import java.util.Treeset;

class Ex
 {
   main ()
   {
   Treeset TS = new TreeSet();
   // TS.add(null); Exception
    TS.add (50);
     TS.add(10);
      TS.add(30);
      TS.add(20);
      TS.add(40);
      S.O.P (TS);
  }
}  // [10, 20, 30, 40, 50]

TS.add( 10 );
TS.add (10);
S.O.P(TS)  // [10].

TS.add(null); // Null Pointer Exception
TS.add("sheela"); // Class Cast Exception.
```

classmate
Date
Page

Queue

java.util.Queue

1. It is an interface defined in java.util Package

1. we have 2 child implementing clause of Queue interface
   - i) PriorityQueue (class)
   - ii) ArrayDeque (class)

3. It stores ~~to~~ object in First In First out (FIFO) order

Characteristics,

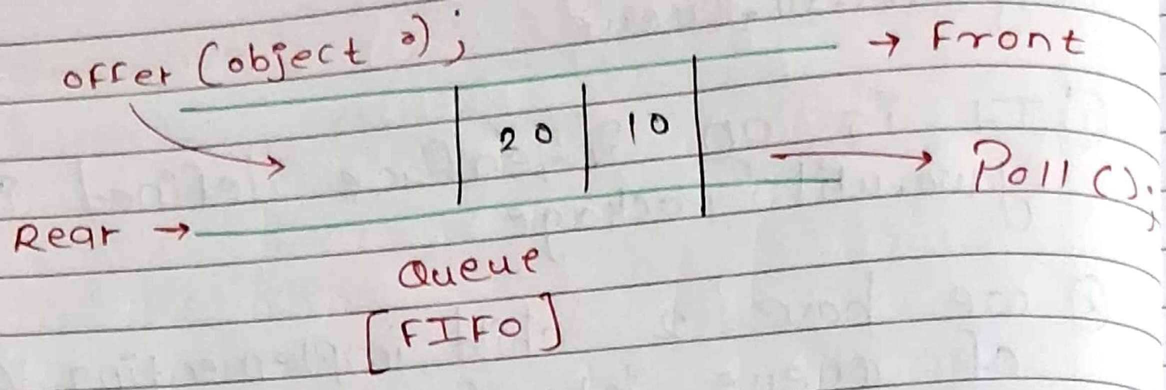
1. No null is accepted

2. Insertion order isn't maintained

3. Duplicate object is ~~maintained~~ ~~are~~ allowed

3. Heterogenous object are not allowed

4. Indexing isn't possible



* java.util.PriorityQueue

- It is a class defined in java.util Package
- it is a subclass of Queue interface

Characteristics

① No Null is accepted

② Insertion order isn't maintained

③ Heterogeneous object aren't allowed

④ Duplicate object aren't allowed

⑤ Indexing Isn't Possible

Program :-

```
import java.util.PriorityQueue
class Ex
{
    main()
    {
        PriorityQueue PQ = new PriorityQueue();
        PQ.offer(5);
        PQ.offer(1);
        PQ.offer(3);
        PQ.offer(4);
        PQ.offer(2);
        S.O.P(PQ);
    }
}
```

Output :-

[1, 2, 3, 5, 4]

classmate
Date
Page

Map

* java.util.Map

① Map is an interface in java which is used to store the objects in the form of key & value pair

② In map, key & value both are java.lang.Object type.

③ Map has 3 child implementing classes to store the object in key-value pair

④ In map, 1 (one) key & value pair is called as entry

Method in Map interface

~~java.uti~~

① adding	put (key, value); PutAll (Map m);
② Removing	clear(); Remove(~~o~~ key); RemoveAll(~~o~~ key, value);
③ Accessing	get(key);
④ searching	Containskey(key); } boolean ContainsValue (value); } entrySet(); key set ();

Entryset = Map gives Result in { } curly braces
keyset = if we Print only keys the result in [ ] views [set form]

classmate
Date
Page

* java.util.HashMap :

HashMap

1. it is subclass of Map Interface
2. It is defined in java.util Package

characteristics HashMap

1. Only one ~~key~~ Null is accepted as a key
2. Insertion isn't maintained
3. Duplicate keys aren't allowed (if we have duplicate keys last value of that will be overridden)
4. Heterogeneous keys are allowed
5. Indexing isn't Possible.

Object: key | value

Program -

```
import java.util.HashMap;
Class Demo {
  main()
  {
   HashMap Hm = new HashMap();
   Hm.put(1,"Sheela");
   Hm.put(2,"Laila");
   Hm.put(null,"Maala");
   Hm.put(2,"Leela");
   S.o.p(Hm);
  }
}   //{null=Maala, 1=Sheela, 2=Leela}
```

classmate
Date
Page

LinkedHashMap

* Java.util.LinkedHashMap;

it is a subclass of Map defined in Java.util Package.

Characteristics

Null is accepted (only one)

Insertion order is maintained

Duplicate keys are not allowed

Heterogeneous keys are allowed

Indexing is not possible

Example -

```
import java.util.LinkedHashMap;

class Ex {
    main() {
    LinkedHashMap LHM = new LinkedHashMap()
        LHm.Put(5, "SHEELA");
        LHM.Put (1, "laila");
        LHM.Put (3, "Malla");
        S.O.p (LHM);
    }
}
```

{ 5 =SHEELA, 1 = laila, 3 = Malla }

classmate

Date

Page

TreeMap

* java.util.TreeMap -

- subclass of Map Interafe
- defined in Java.util Package.

Characteristics

1. No null is accepted as a key.
2. Insertion Order is maintained (ascending)
3. Duplicate object (key) are not allowed
4. Heterogeneous key are not allowed
5. Indexing is not Possible.

Example :-

```
import java.util.TreeMap;
class example {
  main() {
    TreeMap TM = new TreeMap();
    TM.Put(5, "sheela");
    TM.Put(1, 'a');
    TM.Put(3, 100);
    TM.Put(4, 3.14);
    TM.Put(2, new Product());
    S.O.P(tm);
  }
}
// { 1=a, 2=Product@49097b5d, 3=100, 4=3.14, 5=sheela}
```

Exception Handling

Exception — When an error occurs within a method, the method create an object & hand off to the runtime system = Exception object

① It is ab abnormal situation that arises during execution of program & will the stop the program Abroptly

② In Java, Exception is a class

* Exception hierarchy *

(class) Throwable (Superclass of all the Subclass of Exception class)

↓

(class) Exception

- (class) RuntimeException
  - ① → ArithmeticException
  - ② → IndexOutOfBoundsException
    - → ArrayIndexOutOfBoundException
    - → StringIndexoutofBoundException
  - ③ → ClassCastException
  - ④ → NullPointerException
  - ⑤ → NegativeArraySizeException
  - ⑥ → NoSuchElementException
- IOException (class)
- SQLException (class)
- AWTException class

10/0 → Risky code.

* Throwable
java.lang.Throwable

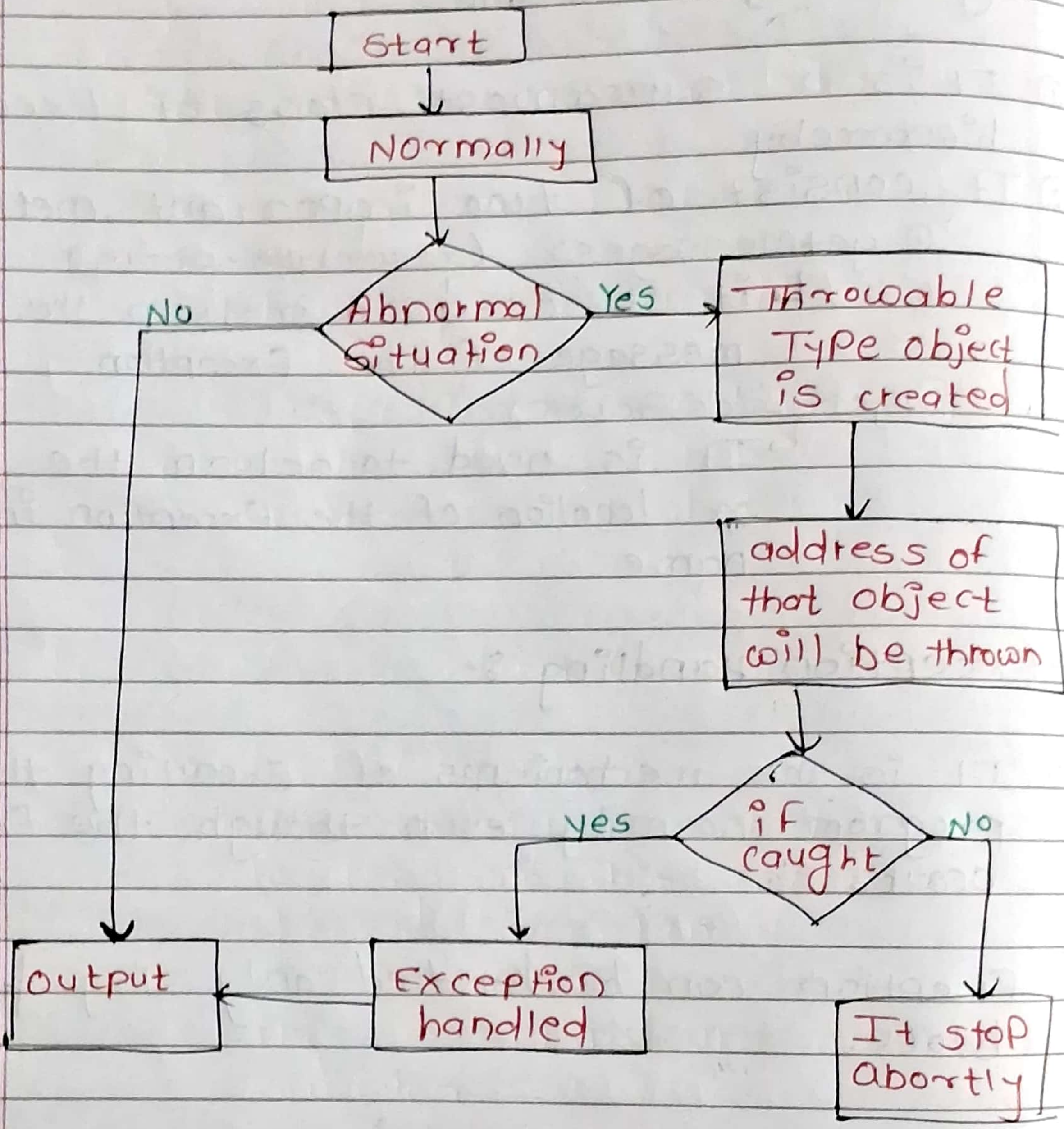
1. It is a supermost class of Exception hierarchy
2. It consist of two important methods
   1. getMessage() (returntype - string)
      ↳ it is used to return the detail message of the Exception
   2. getStackTrace()
      ↳ It is used to return the stackTrace or location of the Exception including name

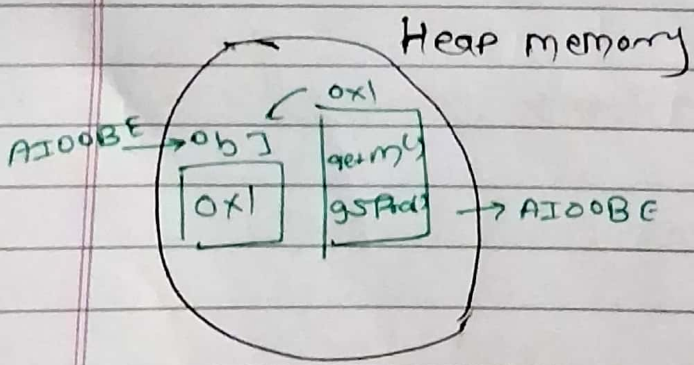
* Exception Handling :-

1. It is the mechanisms of Executing the program normaly even though the Exception occurs.

2. Exception can be handled only Try & catch block.

Q) what happen when Exceptions occures.





classmate
Date
Page

* Try & catch block

```
try {
    // Risky code
}
catch (Throwable var)
{
    // Instruction when exception occur
}
```

Example:-

```
class Test
{
  main()
  {
    S.O.P("Main start");
    try {
        S.O.P(100/0);
    }
    catch (ArithmeticException obj)
    {
        S.O.P(Invalid Denominator");
    }
    S.O.P("Main end");
  }
}
```

⇒ Main start
Invalid Denomimator
Main end

classmate
Date
Page

Example 2 :-

```
class Demo {
  main () {
    int arr[] = new int[5];
    try{
      S.o.p(arr[6]);
    }
    catch(ArrayIndexOutOfBoundException obj)
    {
      S.o.p("Index is out of Range");
    }
  } => Index is out of Range
}
```

* Types of Exception

i) Checked Exception

- the exception which is known to the compiler is called as checked exception

  OR

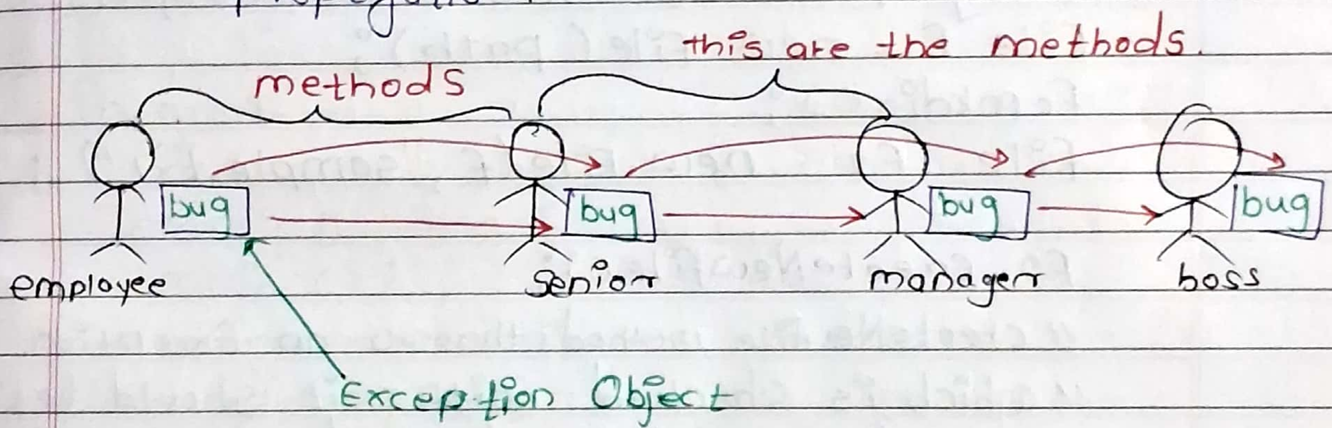
  this exceptions are aware to the compiler

- checked exceptions ~~as~~ occures compile time

ii) Unchecked Exception -
the exception which is unknown to the compiler is called as unchecked Exception

Examples :-
All runtime Exception & its child.

iii) Exception object Propogation -
the movement of one exception object frome the called method to the calling method when it is not handled is called exception object propogation.



Throws :-
1. it is a keyword
2. it is used to declare the exception
3. it can be used with only method declaration
4. we can declare multiple exception at a time.

NOTE :- 1. we can handle the exception by using Try & catch block

Date
Page

② checked exception can handled or they can be declared caller also using throws keyword.

Syntax :-
method declaration throws Except^{n}1, Except^{n}2 ....{
}

Example :-

```
class Demo{
  main() throws IOException{
  String path = "C:\\users\\Desktop\\Objects";
  File f = new File(path);
  f.mkdir();
  File f2 = new File(f, "sample.txt");

  f2.createNewFile();
  // createNewFile method throws an Exception
  // which is checked ~~with~~ so it should be either
  // handled or should be declared using throws
  }
}
```

classmate
Date
Page

(iii) Custom Exception :-

- The exception created by the programmer is known as custom exception
Steps to custom exception -

Step - ① Create a class & make it child of throwable.

Step - ② Override ~~get message()~~ getMessage() & Print relevant message

Step - ③ throw the exception using throw keyword.

Program :-

```
class Demo(){
 main (){
    int age = 18;
    if (age<18) {
        throw new underAgeException();
    }
    elese
        S.o.p("Eligible to vote !!");
 }
}
class underAgeException extends RuntimeException{
 public String getMessage() {
      return "person is under age";
 }
} // Eligible to vote !!.
```

throws	throw
① it is used to declare the exception	It is used to throw the exception object
② throws can be used only in the method declaration	throw can be used anywhere in the program

NOTE :-

① If a custom Exception Extends checked exception it becomes type of checked exception.

② If a custom Exception extends unchecked exception it becomes an unchecked exception

Finally :-

① it is a block
② it used with try & catch block.
③ it is used for cleanup activities or closing the costly resource.
④ Finally block will execute if exception occurs or if does not occurs. it is independent of exception.